Graduation Final Work to qualify for the
Bachelor's degree in Computer Engineering

DEVELOPMENT OF A SOFTWARE APPLICATION FOR THE EXECUTION OF THE QUANTITATIVE ASSOCIATION RULES MINING ALGORITHM (QUARG)

Intelligent Data Analysis Research Lab
Czech Technical University in Prague

Written by Adalberto Cubillo Rojas

School of Computer Engineering
Costa Rica Institute of Technology

Faculty adviser: Santiago Nuñez Corrales

Prague, January 2012

Executive Summary

Nowadays, the use of the information and communication technologies has resulted in storage of large amounts of data every day. That data can be used later as a source of knowledge to enterprises and organizations through the use of data mining techniques.

One of the most frequently applied data-mining techniques is association rules. This is due of their ability to identify interesting relationships among huge amounts of data. But for certain types of data it often results in a computationally expensive algorithm.

According to F. Karel [3], the association rules are understood as an implication $X \rightarrow Y$ in a transactional database, each of which contains a subset of items or attributes, where the item sets X and Y are disjoint. The left side of this implication is called the *antecedent*, and the right side is referred to as *consequent*. An example of this is: having the statement “if a customer buys a toothbrush, then she also probably buys toothpaste” then the association rule can be written as: $\{\text{toothbrush}\} \rightarrow \{\text{toothpaste}\}$.

In 2009, Filip Karel proposed and verified in his doctoral thesis an efficient way to obtain this knowledge using ordinal association rules. He developed an alternative approach to the quantitative association rule mining (QUARG).

This document intends to present the QUARGsoft application, which main objective is to develop an efficient implementation of the quantitative association rules mining algorithm (QUARG) into a web application. The document starts introducing the description of the problem with the details of how works the QUARG genetic algorithm and the role of the Intelligent Data Analysis Research Laboratory within this project.

Then, the document introduces the generalities of the project and shows how the problem was solved using open source tools during the development of it, within the important elements are the taken design, the developed pseudo-code, and the source code of the application. All of this let the reader to have an comprehensive view of the application's functionalities.

Finally, some runtime comparison results, between the experimental scripts in Matlab and the developed application in Java, are shown. Which denote the accomplishment of the project's main objective and concludes that the QUARGsoft application improves the Filip Karel's experimental scripts reducing the runtime of the algorithm more than 50%.

Keywords: Data mining, Java, Web application, Performance.

Contents

List of Tables

List of Figures

1. Description of the Problem

1.1. Enterprise Description

The Czech Technical University in Prague was funded in 1707 by the emperor Joseph 1, being nowadays one of the biggest and oldest technical universities in Central Europe [1].

The university counts with eight faculties: civil engineering, mechanical engineering, electrical engineering, nuclear sciences and physical engineering, architecture, transportation sciences, biomedical engineering and information technology.

This university has graduated a large quantity of world-renowned scientists and winners of important awards by their contributions to the sciences and developments impacting the contemporary world.

One of the main laboratories from the cybernetics department of the electrical engineering faculty is the Intelligent Data Analysis Research Lab. Its principal function is to accomplish that computers can discover knowledge in data automatically [4].

The work in this laboratory is focused on development of algorithms for data-mining and machine learning. The purpose of this is to detect irregularities and patterns that lead to novel information, build predictive models and identify the results from the analyzed data. Among the principal areas of algorithm implementation are bioinformatics and intelligent optimization.

1.2. QUARG Algorithm Detailed Description

1.2.1 Data Preprocessing

Preprocessing the data input in the QUARG algorithm is important because its main objective is to reduce a potentially infinite number of values of quantitative attributes. This process consist first to discretize the input values and secondly mapping them.

Some of the advantages of using discrete values rather than continuous ones are:

- Discrete features are closer to an knowledge-level representation.
- This kind of values are easier to understand.
- Discretization makes learning more accurate and faster.

For the QUARG algorithm the selected discretization method was K-means algorithm, it is an automatic unsupervised method which adapts to data's character and combine advantages of Equi-Depth (equally as many records is assigned to each discretization bin) and Equi-Distance (each discretization bin has the same range of values) algorithms.

Then, the resulting values from discretization are mapping to consecutive rows of integers beginning with 1 (this value represents the lowest value of an atomic attribute), the combination of both processes, discretization and mapping covers the fact that F. Karel [3] presents in his analysis “distance among particular records in 2-D space is important during quantitative rules mining”, which develop an optimization when running the algorithm.

1.2.2. Construction of the Rules

An association rule is understood as an implication $X \rightarrow Y$ where the left side of this implication is called antecedent and the right side is called succedent. Both of these, antecedent and succedent, consist in a combination of one or more preprocessed atomic attributes (represented them by one vector), and there is no limitation in the number of attributes for each part, only limited by the user according specific needs in the analysis.

Also, these rules are generated avoiding analyze redundant rules. The QUARG algorithm searches and selected the follow rules patterns:

- Antecedent and succedent cannot have repeated atomic attributes, for example is valid a rule $X \rightarrow Y$ but is not valid a rule like $Y \rightarrow Y$ or $X \wedge Y \rightarrow Y$.
- The rules $X \rightarrow Y$ and $Y \rightarrow X$ are considered different and both are verified by the QUARG algorithm.
- In the case of having the rules $X \wedge Y \rightarrow Z$ and $Y \wedge X \rightarrow Z$ the algorithm choose only one of them.

1.2.3. Areas of Interest Identification

The areas of interest are areas of strong association between antecedent and succedent. Instead of searching in the whole space of relationship, the QUARG algorithm focuses only on the selected areas of interest causing a reduction in the searching space and consequently in a reduction on the time consumption.

Therefore is essential to identify the best areas of interest to obtain the best candidate rules. F. Karel [3] proposes the combination of the chi-square tests approach (test of independence) and genetic algorithms.

For the test of independence each compound attribute has a discrete distribution (described as a vector) and a joint distribution of two compound attributes (described as a matrix) that indicates the amount of records being analyzed for a certain rule. And the algorithm uses two types of joint distribution [3]:

1. Real joint distribution: $P_{xy}(t,s)$, where the value v_{ts} in the point $[t,s]$ of the matrix is the real value corresponding to the number of records in the data input.

2. Joint distribution under assumption of independence: $P_x(t)P_y(s)$, where the value e_{ts} int the point $[t,s]$ is theoretically estimated using particular real distributions $P_x(t)$ and $P_y(s)$ and assuming their independence, this value is given by:

$$e_{ts} = \frac{v_t * v_s}{M}$$

where v_t is the value of the real distribution $P_x(t)$ in the point $[t]$, v_s is the value of the real distribution $P_y(s)$ in the point $[s]$ and M is the total number of records in the data input.

And P is the difference matrix which the sum of all its values is 0

$$P = P_{xy}(t,s) - P_x(t)P_y(s)$$

The strongest associations between antecedent and succedent are located in areas with high positive values, therefore, in areas where the difference between real and theoretical distribution of the compound is highest.

Finally, the genetic algorithm takes place. This algorithm is based on the principles of evolution via natural selection, employing a population of individuals and applying to it operators such as mutation, recombination and fitness.

For the task of areas of interest identification every individual in the population is represented by four integers: $i(p)=[t_{min}, s_{min}, t_{max}, s_{max}]$ and these values represent an specific area of interest. The genetic algorithm works as follow [3]:

1. Generate randomly the population of individuals.
2. Compute fitness function for each individual.
3. Apply selection.
4. Apply crossover.
5. Mutate population.
6. Complete population by randomly generated individuals.
7. Repeat from step 2 until the number of generations is reached.
8. Select the final areas of interest.

1.2.4. Areas of Interest Decomposition

Each area of interest is determined by the following parameters: $t_{min}, s_{min}, t_{max}, s_{max}$ the concrete values of particular attributes have to be extracted from that coordinates. During the decomposition those values represent a K-D space, where D dimensions are necessary for K atomic attributes present in a compound attribute for the antecedent and succedent conditions [3].

To generate these conditions the algorithm generates all the squares and rectangles from the area, which bounders the solution. And by decomposing the antecedent part of the area of interest the solution is the list of antecedent conditions, same for the succedent.

1.2.5. Verification of Candidate Rules

Once the algorithm have the list of candidate rules it have to verified the validity of them, only those with that meets the interestingness thresholds are selected for the result output.

The interestingness thresholds applied in the algorithm are: confidence, support and lift. To explain each of them is necessary denote the following parameters:

- ant – the number of records in the data input that satisfy the antecedent condition of the rule that is being evaluated.
- suc – the number of records in the data input that satisfy the succedent condition of the rule that is being evaluated.
- ant_suc – the number of records in the data input which satisfy both (antecedent and succedent) conditions of the candidate rule.
- M – the total number of records in the data input.

Confidence is a measure of rule's reliability, tells how much the rule can be trusted, and the value of it lies in the interval of <0,1>. Confidence is given by

$$conf = ant\_suc / ant$$

Support tells how frequent the rule is, and the value of it lies in the interval of <0,1>. Support is given by

$$supp = ant\_suc / M$$

Lift measures the performance of the algorithm at predicting the resulting rule, the value of it lies in the interval of <0, ∞ >. Lift is given by

$$lift = ant\_suc * M / ant * suc$$

Finally, the algorithm select the result rules validating all the three measures and comparing those with the input values of the minimum confidence, support and lift and that way remove the weak rules.

1.3. Objectives of the Project

1.3.1. Main Objective

To develop an efficient and general application of the QUARG algorithm, proposed and experimentally tested by Filip Karel, to obtain association rules from ordinal data.

1.3.2. Specific Objectives

- To standardize the input and output data format for the application.
- To test the performance of the application and compare it against the experimental scripts developed on MATLAB.
- To elaborate a user guide for the application and documentation of the source code for future references or extensions of the functionality of the program.

1.4. Expected Products

- An optimized, general implementation of the experimental scripts developed by Filip Karel.
- Test results and their comparison against those from the experimental scripts.
- An user guide, description of program functionality and source code documentation.

1.5. Scopes and Limitations

1.5.1. Scopes

1. The current project only considers the existing theory and the scripts on which the project is originally based.

2. The definition of new functionality seeks to accomplish a standardization and optimization of the code.

3. A web service and a website will be developed in order to provide access to the application for the community of potential users.

4. Tests of the new functionality will benchmark against metrics between computer resource usage of the new code and the current experimental scripts.

5. Write an user guide and a detailed documentation of the new code implemented.

1.5.2. Limitations

1. The project will not include any security aspects related to data transmission through the Internet.

1.6. Technologies and Methodology

The methodology of development on the project is the incremental approach, which subdivide the project in small segments defined by the important parts of the theoretical basis: rules creation, areas of interest identification, areas of interest decomposition, and rule verification.

This subdivision will allow to generate specific optimization test results, reduce the project risk and better bug fixing. Also simplifies the process of adding new components to the application, and making changes according to the researcher needs.

The technologies selected for the application development are:

Java, the application functionalities lie on a web service based on this object-oriented language, allowing a better functionality structure and a the application modularization.

HTML and JQuery, the graphical user interface is developed on these technologies, HTML provides a perfect distribution route of the application through the Internet, and the use of JQuery simplifies the connection with the web service and the creation of an user friendly application.

2. Implemented Solution

2.1. Description of the Project Solution

The project consisted on the development of an web application that implemented the QUARG algorithm. And was developed using HTML, CSS, Flash and JQuery in a way that the graphical interface became user-friendly.

Between the client layer (user interface) and the logic layer was implemented PHP as a controller layer allowing the application to connect with a Java Web Service as the logical layer of the application, where all the methods and functionality were implemented.

The application receives a file with the data input values in the coma separated format (CSV), where the first row is the attributes list (e.g. sugar;milk;bread) and the rest of the rows are the records (e.g. 5;23;4), one record per row.

Then the user set the options for executing the preprocessing algorithm (if necessary to discretize the input data). After that, the application shows the settings for applying the rules mining algorithms, and the option the execute it.

Finally, the system returns a list with all the valid rules generated by the QUARG algorithm. The user has the option of saving the results in a text file, and the option of saving the preprocessed data in another CSV file.

All the functions and methods are going to be standardized and divided into modules, so that more functionality can be added in a simpler form to the application without having to make large changes into the code.

The implementation must have better performance than the experimental scripts developed by Filip Karel, allowing it to solve quantitative associations with a higher complexity (several attributes combinations in antecedent and/or succedent) or size (large amount of data records). This is going to be analyzed through testing the difference on the runtime between the experimental scripts and the program.

Also, a terminal version of the application was develop with the purpose of allowing the researcher to run the program locally.

Finally, the whole program functionality and the methods will be documented for future references and changes, including a user manual of the application.

2.2. Design

2.2.1. Classes Diagram

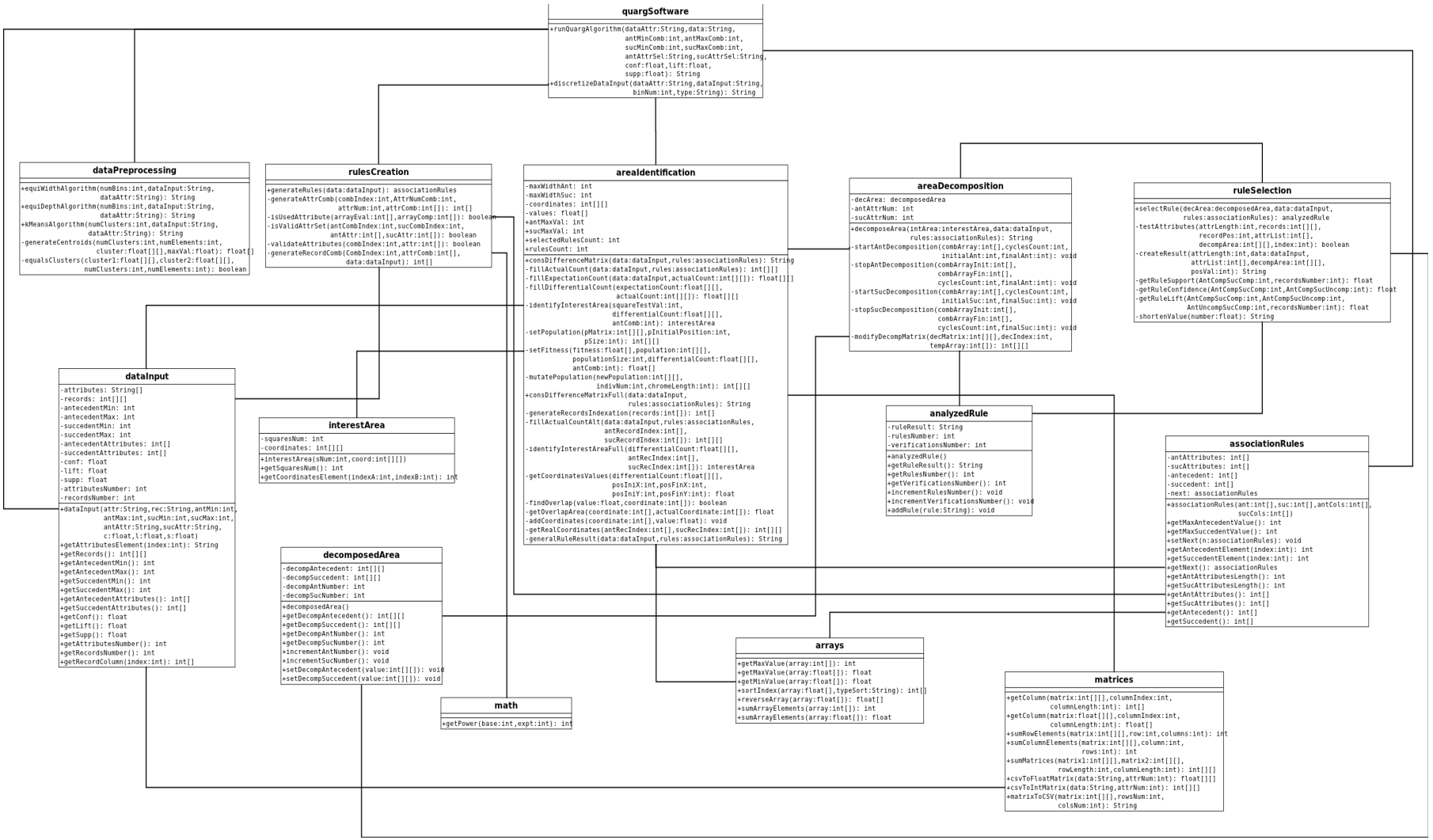

Figure 1. Project's classes definition diagram.

2.2.2. Components Diagram

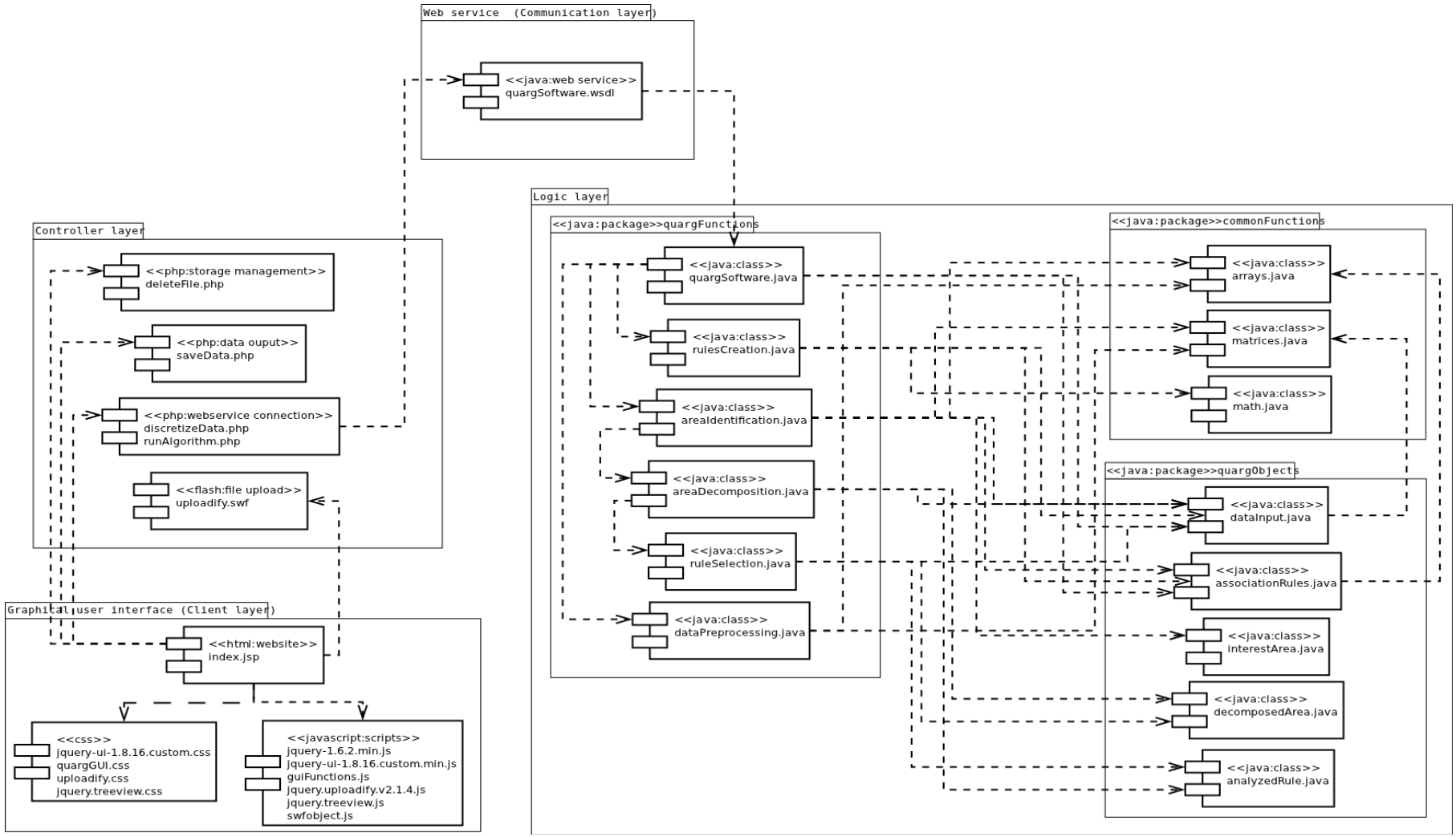

Figure 2. Interaction diagram of project components.

2.2.3. Sequence Diagram

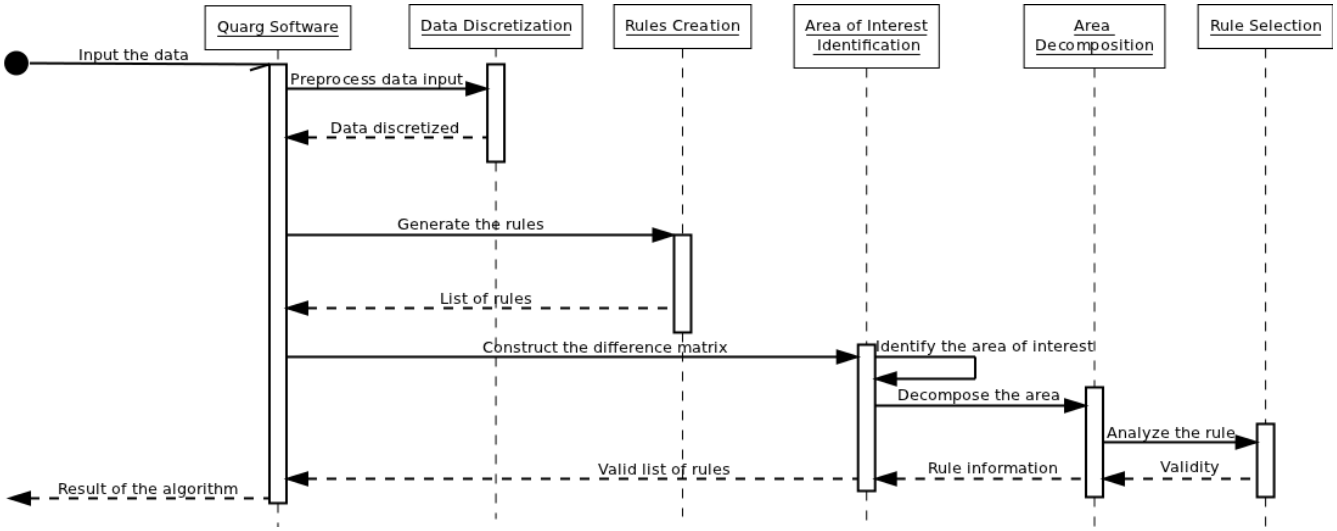

Figure 3. Diagram for the processes sequence in the project.

2.2.4. Activities Diagrams

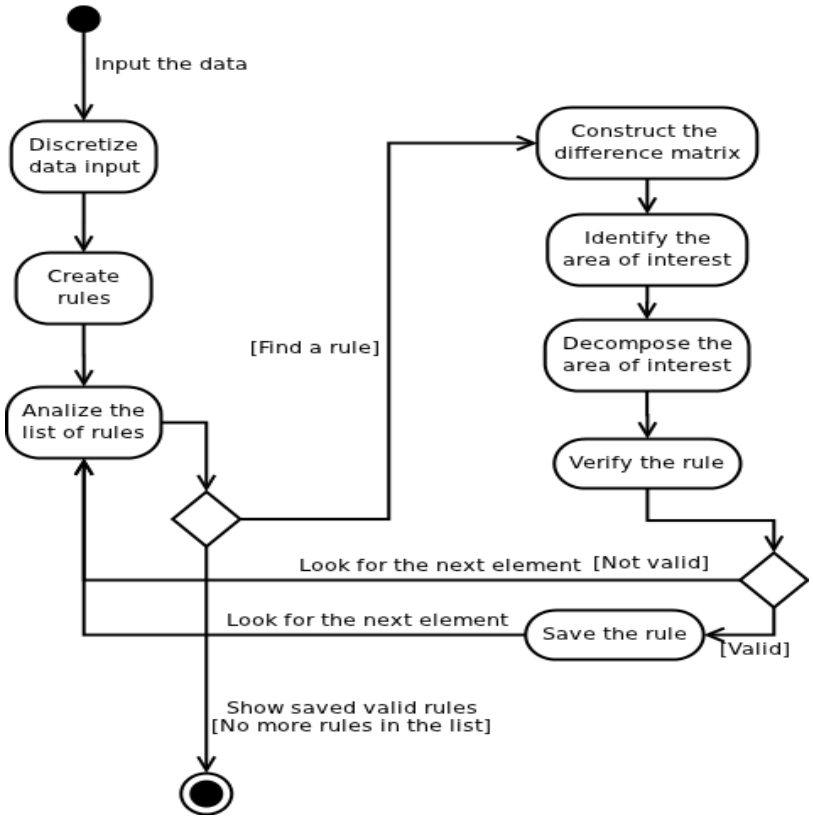

Figure 4. Overall activity diagram.

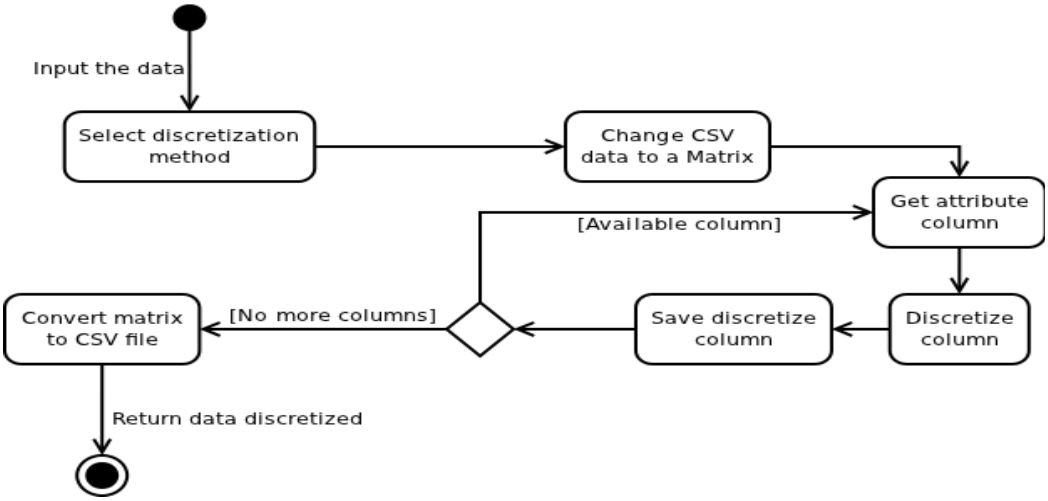

Figure 5. Data input preprocessing (discretization) activity diagram.

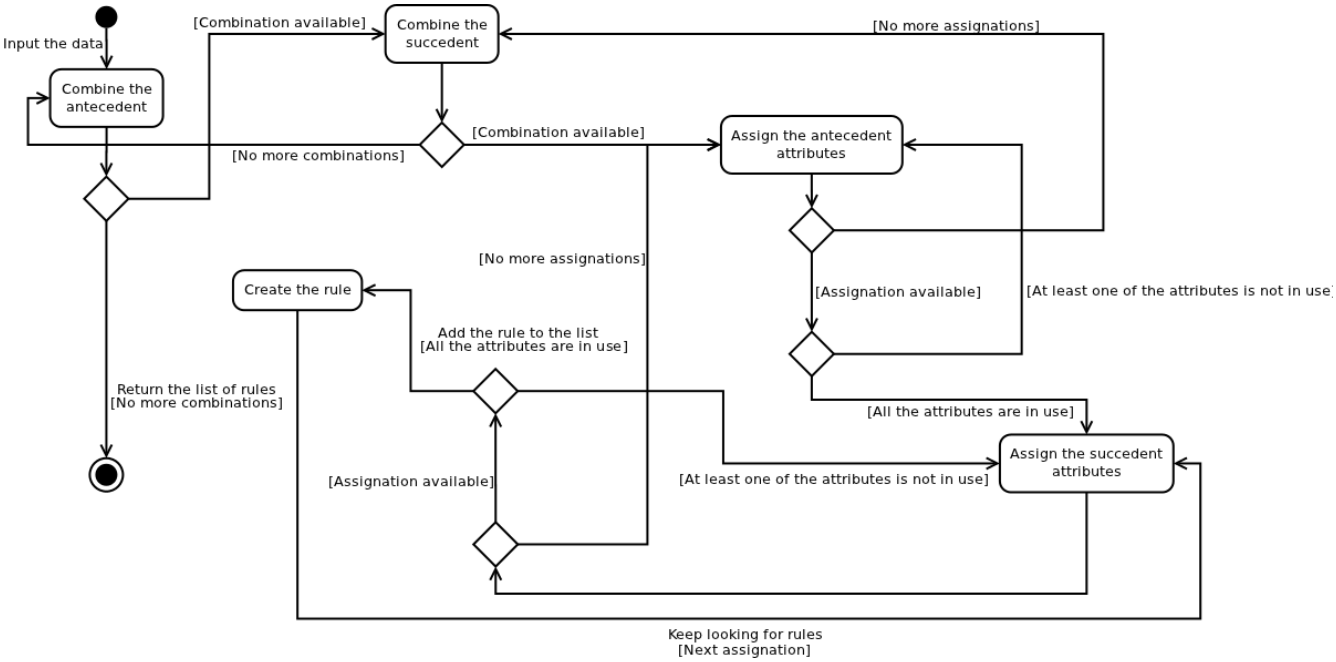

Figure 6. Rules creation activity diagram.

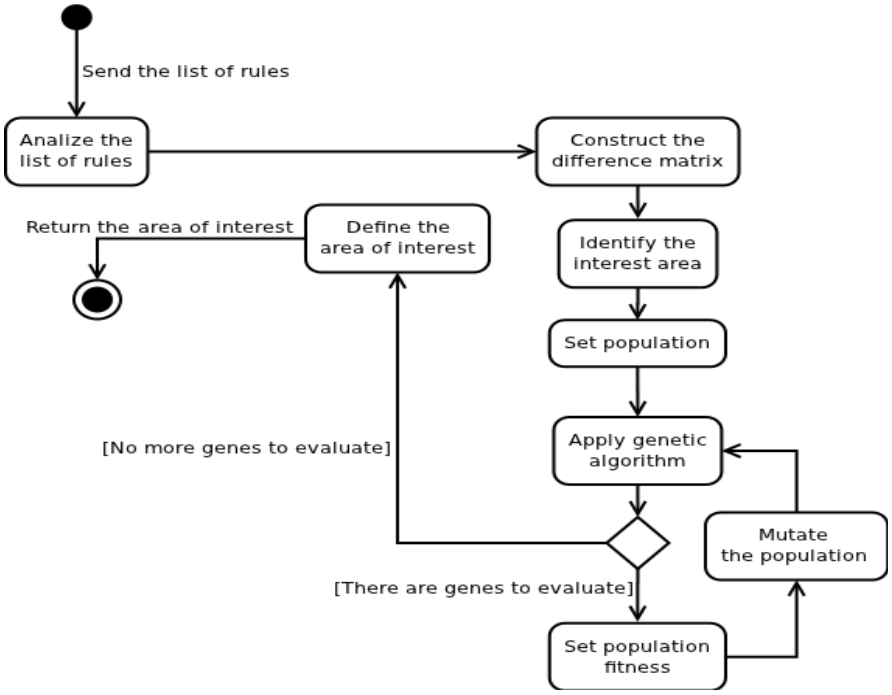

Figure 7. Areas of interest identification (genetic algorithm) activity diagram.

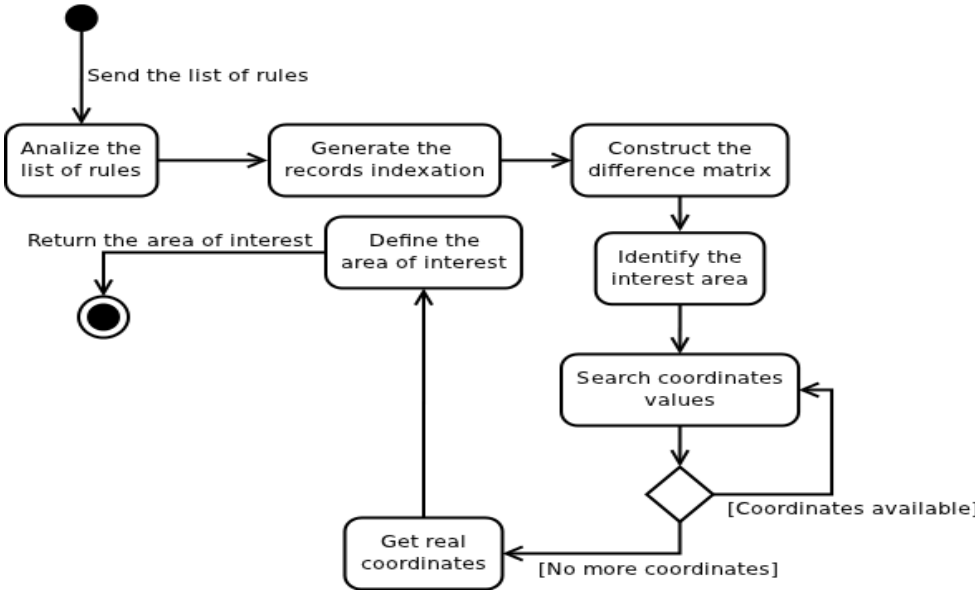

Figure 8. Areas of interest identification (full search algorithm) activity diagram.

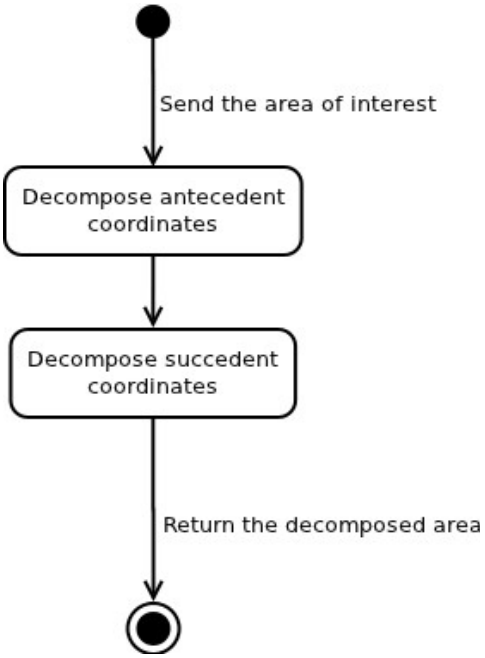

Figure 9. Areas of interest decomposition activity diagram.

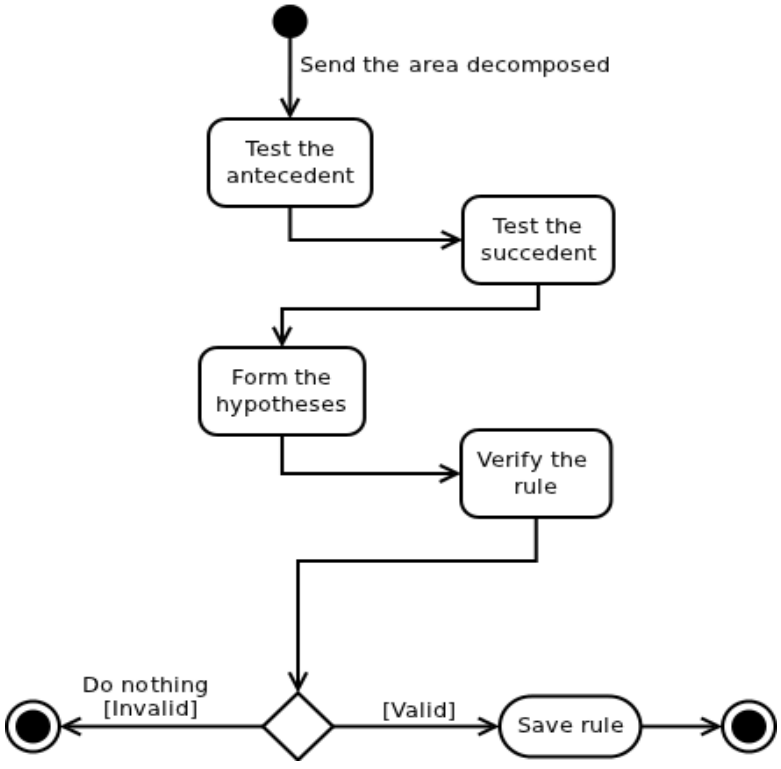

Figure 10. Rule verification activity diagram.

2.3. Algorithms Pseudo-code

2.3.1. commonFunctions Package

2.3.1.1. arrays.java Class

Table 1. Pseudo-code for getting the maximum value from an array of integer.

public static int **getMaxValue**(int[] array)

begin
  set *max* = 0
  for *i* = 0 **to** *array*.length
  if *array*[*i*] > *max* **then**
  set *max* = *array*[*i*]
  end if
  return *max*
end

Table 2. Pseudo-code for getting the maximum value from an array of float.

public static float **getMaxValue**(float[] array)

begin
  set *max* = 0
  for *i* = 0 **to** *array*.length
  if *array*[*i*] > *max* **then**
  set *max* = *array*[*i*]
  end if
  return *max*
end

Table 3. Pseudo-code for getting the minimum value from an array of float.

public static float **getMixValue**(float[] array)

begin
  set *min* = 0
  for *i* = 0 **to** *array*.length
  if *array*[*i*] < *min* **then**
  set *min* = *array*[*i*]
  end if
  return *min*
end

Table 4. Pseudo-code for sorting the index of the array.

public static int[] **sortIndex**(float[] array, String typeSort)
begin   **set** *fitnessCopy* = **call** Arrays.copyOf with *array*, *array*.length

```
  set arraySorted = call Arrays.copyOf with array, array.length
  initialize index = new int[array.length]
  call Arrays.sort with arraySorted
  if typeSort is equal to “descend” value then
    set arraySorted = call reverseArray with arraySorted
  end if
  for i = 0 to arraySorted.length
    set pos = 0
    for j = 0 to fitnessCopy.length
      if fitnessCopy[j] != -50000 and arraySorted[i] == fitnessCopy[j] then
        set fitnessCopy[j] = -50000
        break
      else
        increment pos
      end if
    set index[i] = pos
  return index
end
```

Table 5. Pseudo-code for reversing an array.

```
public static float[] reverseArray(float[] array)
begin
  initialize reversed = new float[array.length]
  for i = 0 to array.length
    set reversed[i] = array[array.length - i - 1]
  return reversed
end
```

Table 6. Pseudo-code for summing the integer array elements.

```
public static int sumArrayElements(int[] array)
begin
  set result = 0
  for index = 0 to array.length
    set result = result + array[index]
  return result
end
```

Table 7. Pseudo-code for summing the float array elements.

```
public static int sumArrayElements(float[] array)
begin
  set result = 0
  for index = 0 to array.length
    set result = result + array[index]
```

```
  return result
end
```

2.3.1.2. math.java Class

Table 8. Pseudo-code for getting the value of the power.

```
public static int getPower(int base, int expt)
begin
  set result = 1
  for index = 0 to expt
    set result = result * base
  return result
end
```

2.3.1.3. matrices.java Class

Table 9. Pseudo-code for getting a specific matrix integer column.

```
public static int[] getColumn(int[][] matrix, int columnIndex, int columnLength)
begin
  initialize column = new int[columnLength]
  for i = 0 to columnLength
    set column[i] = matrix[i][columnIndex]
  return column
end
```

Table 10. Pseudo-code for getting a specific matrix float column.

```
public static float[] getColumn(float[][] matrix, int columnIndex, int columnLength)
begin
  initialize column = new float[columnLength]
  for i = 0 to columnLength
    set column[i] = matrix[i][columnIndex]
  return column
end
```

Table 11. Pseudo-code for summing the row elements of a matrix.

```
public static int sumRowElements(int[][] matrix, int row, int columns)
begin
  for i = 0 to columns
    set result += matrix[row][i]
  return result
end
```

Table 12. Pseudo-code for summing the column elements of a matrix.

public static int **sumColumnElements**(int[][] matrix, int column, int rows)

begin
  for *i* = 0 **to** *rows*
  set *result* += *matrix*[*i*][*column*]
  return *result*
end

Table 13. Pseudo-code for summing the elements of two matrices.

public static int[][] **sumMatrices**(int[][] matrix1, int[][] matrix2, int rowLength, int columnLength)

begin
  initialize *result* = **new** int[*rowLength*][*columnLength*]
  for *x* = 0 **to** *rowLength*
  for *y* = 0 **to** *columnLength*
  set *result*[*x*][*y*] = *matrix1*[*x*][*y*] + *matrix2*[*x*][*y*]
  return *result*
end

Table 14. Pseudo-code for converting a CSV string to a float matrix.

public static float[][] **csvToFloatMatrix**(String data, int attrNum)

begin
  set *dataRows* = **call** *data*.split with ","
  set *dataRowsLength* = *dataRows*.length
  initialize *records* = **new** float[*dataRowsLength*][*attrNum*]
  for *y* = 0 **to** *dataRowsLength*
  set *rowData* = **call** *dataRows*[*y*].split with ";"
  for *x* = 0 **to** *attrNum*
  set *records*[*y*][*x*] = **call** Float.valueOf with *rowData*[*x*]
  return *records*
end

Table 15. Pseudo-code for converting a CSV string to a integer matrix.

public static int[][] **csvToIntMatrix**(String data, int attrNum)

begin
  set *dataRows* = **call** *data*.split with ","
  set *dataRowsLength* = *dataRows*.length
  initialize *records* = **new** int[*dataRowsLength*][*attrNum*]
  for *y* = 0 **to** *dataRowsLength*
  set *rowData* = **call** *dataRows*[*y*].split with ";"
  for *x* = 0 **to** *attrNum*
  set *records*[*y*][*x*] = **call** Integer.valueOf with *rowData*[*x*]
  return *records*
end

Table 16. Pseudo-code for converting a integer matrix into a CSV string format.

public static String **matrixToCSV**(int[][] matrix, int rowsNum, int colsNum)

begin
  initialize *csv* = **new** StringBuilder with 10000
  for *r* = 0 **to** *rowsNum*
  for *c* = 0 **to** *colsNum*
  call *csv*.append with *matrix*[*r*][*c*]
  if *c* != (*colsNum* – 1) **then**
  call *csv*.append with ';'
  end if
  call *csv*.append with “\n”
  return (**call** *csv*.toString)
end

2.3.2. quargFunctions Package

2.3.2.1. quargSoftware.java Class

Table 17. Pseudo-code for the rules mining algorithm main function.

public String **runQuargAlgorithm**(String dataAttr, String data, int antMinComb, int antMaxComb, int sucMinComb, int sucMaxComb, String antAttrSel, String sucAttrSel, float conf, float lift, float supp, String algorithm)

begin
  try
  initialize *evaluationData* = **new call** dataInput constructor with *dataAttr*, *data*, *antMinComb*, *antMaxComb*, *sucMinComb*, *sucMaxComb*, *antAttrSel*, *sucAttrSel*, *conf*, *lift*, *supp*
  catch on *exception*
  return "There is a problem with the input data, please check it."
  try
  set *generatedRules* = **call** generateRules with *evaluationData*
  if *algorithm* equals to “genetic” **then**
  set *result* = **call** consDifferenceMatrixQUARG with *evaluationData*, *generatedRules*
  else if *algorithm* equals to “full” **then**
  set *result* = **call** consDifferenceMatrixFull with *evaluationData*, *generatedRules*
  else
  return "You have selected an algorithm not implemented yet."
  end if
  return *result*
  catch on *exception*
  return "Something wrong happened, check the code." + *exception*
end

Table 18. Pseudo-code for the preprocessing algorithm main function.

public String **discretizeDataInput**(String dataAttr, String dataInput, int binNum, String type)

```
begin
  if type equals to “width” then
    try
      return (call equiWidthAlgorithm with binNum, dataInput, dataAttr)
    catch on exception
      return "Something wrong happened, check the code!!" + exception
  else if type equals to “depth” then
    try
      return (call equiDepthAlgorithm with binNum, dataInput, dataAttr)
    catch on exception
      return "Something wrong happened, check the code!!" + exception
  else if type equals to “kmeans” then
    try
      return (call kMeansAlgorithm with binNum, dataInput, dataAttr)
    catch on exception
      return "Something wrong happened, check the code!!" + exception
  else
    return "You have selected an algorithm not implemented yet."
  end if
end
```

2.3.2.2. dataPreprocessing,java Class

Table 19. Pseudo-code for discretize the input data using the Equi-Width Algorithm.

public static String **equiWidthAlgorithm**(int numBins, String dataInput, String dataAttr)

```
begin
  set numAttr = (call dataAttr.split with “;”).length
  set data = call csvToFloatMatrix with dataInput, numAttr
  set dataLength = data.length
  initialize bins = new float[numBins]
  initialize result = new StringBuilder with 10000
  initialize dataDiscretized = new int[dataLength][numAttr]
  for i = 0 to numAttr
    set column = call getColumn with data, i, dataLength
    set minVal = call getMinValue with column
    set maxVal = call getMaxValue with column
    set size = (maxVal – minVal) / numBins
    for b = 1 to numBins
      set value = minVal + (size * b)
      set bins[b – 1] = value
    set bins[numBins – 1] = maxVal + 1
    for index = 0 to dataLength
      for b = 0 to numBins
```

```
      if column[index] < bins[b] then
        set dataDiscretized[index][i] = b + 1
        break
      end if
  call result.append with (call matrixToCSV with dataDiscretized, dataLength, numAttr)
  return (call result.toString)
end
```

Table 20. Pseudo-code for discretize the input data using the Equi-Depth Algorithm.

```
public static String equiDepthAlgorithm(int numBins, String dataInput, String dataAttr)
begin
  set numAttr = (call dataAttr.split with ";").length
  set data = call csvToFloatMatrix with dataInput, numAttr
  set dataLength = data.length
  initialize sortedColumn = new float[dataLength]
  initialize bins = new float[numBins]
  initialize binsPositions = new int[numBins]
  initialize result = new StringBuilder with 10000
  initialize dataDiscretized = new int[dataLength][numAttr]
  for b = 1 to numBins
    set value = (dataLength * b) / numBins
    set binsPositions[b – 1] = call Math.round with value
  for i = 0 to numAttr
    set column = call getColumn with data, i, dataLength
    call System.arraycopy with column, 0, sortedColumn, 0, dataLength
    call Arrays.sort with sortedColumn
    for b = 0 to numBins
      set bins[b] = sortedColumn[binsPositions[b] – 1]
    for index = 0 to dataLength
      for b = 0 to numBins
        if column[index] <= bins[b] then
          set dataDiscretized[index][i] = b + 1
          break
        end if
  call result.append with (call matrixToCSV with dataDiscretized, dataLength, numAttr)
  return (call result.toString)
end
```

Table 21. Pseudo-code for discretize the input data using the K-Means Algorithm.

```
public static String kMeansAlgorithm(int numClusters, String dataInput, String dataAttr)
begin
  set numAttr = (call dataAttr.split with ";").length
  set data = call csvToFloatMatrix with dataInput, numAttr
  set dataLength = data.length
```

```
  initialize result = new StringBuilder with 10000
  initialize dataDiscretized = new int[dataLength][numAttr]
  for i = 0 to numAttr
    set column = call getColumn with data, i, dataLength
    set maxVal = call getMaxValue with column
    set centroids = call generateCentroids with numClusters, dataLength, null, maxVal
    set move = true
    set finalClusters = null
    while move
      initialize distance = new float[numClusters][dataLength]
      initialize clusters = new float[numClusters][dataLength]
      for c = 0 to numClusters
        for pos = 0 to dataLength
          set distances[c][pos] = call Math.abs with (column[pos] – centroids[c])
      for pos = 0 to dataLength
        set posMinElement = 0
        for c = 1 to numClusters
          if distances[posMinElement][pos] > distances[c][pos] then
            set posMinElement = c
          end if
        set clusters[posMinElement][pos] = column[pos]
      if (call equalsClusters with finalClusters, clusters, numClusters, dataLength) then
        set move = false
      else
        set centroids = call generateCentroids with numClusters, dataLength, clusters, maxVal
        set finalClusters = clusters
      end if
    for index = 0 to dataLength
      for c = 0 to numClusters
        if finalClusters[c][index] != 0 then
          set dataDiscretized[index][i] = c + 1
        end if
  call result.append with (call matrixToCSV with dataDiscretized, dataLength, numAttr)
  return (call result.toString)
end
```

Table 22. Pseudo-code for generate the centroids for the K-Means algorithm.

```
private static float[] generateCentroids(int numClusters, int numElements, float[][] cluster,
                                         float maxVal)
begin
  initialize centroids = new float[numClusters]
  if clusters == null then
    for i = 0 to numClusters
      set centroids[i] = (call Math.random) * maxVal
  else
```

```
    for c = 0 to numClusters
      set cantElements = 0
      set sumElements = 0
      for i = 0 to numElements
        set element = cluster[c][i]
        if element != 0 then
          increment cantElements
          set sumElements = sumElements + element
        end if
    set centroids[c] = sumElements / cantElements
  end if
  return centroids
end
```

Table 23. Pseudo-code for comparing the clusters in the K-Means algorithm.

```
private static boolean equalsClusters(float[][] clusters1, float[][] clusters2, int numClusters,
                                      int numElements)
begin
  if clusters1 != null then
    for c = 0 to numClusters
      for pos = 0 to numElements
        if clusters1[c][pos] != clusters2[c][pos] then
          return false
        end if
    return true
  else
    return false
  end if
end
```

2.3.2.3. rulesCreation.java Class

Table 24. Pseudo-code for generating the list of the rules to evaluate.

```
public static associationRules generateRules(dataInput data)
begin
  set antAttr = null
  set sucAttr = null
  set result = null

  // Number of antecedent elements combinations.
  for antCombIndex = (call data.getAntecedentMin) to (call data.getAntecedentMax)
    set antAttrCombNum = call Math.pow with (call data.getAttributesNumber), antCombIndex

    // Number of succedent elements combinations.
```

```
  for sucCombIndex = (call data.getSuccedentMin) to (call data.getSuccedentMax)
    set sucAttrCombNum = call Math.pow with (call data.getAttributesNumber),
                                                    sucCombIndex

    // Antecedent attribute number assignations.
    for antAttrComb = 1 to antAttrCombNum
      set antAttr = call generateAttrComb with antCombIndex, antAttrComb,
                                              (call data.getAttributesNumber), antAttr
      if (call isUsedAttribute with antAttr, (call data.getAntecedentAttributes)) == true then

        // Succedent attribute number assignations.
        for sucAttrComb = 1 to sucAttrCombNum
          set sucAttr = call generateAttrComb with sucCombIndex, sucAttrComb,
                                              (call data.getAttributesNumber), sucAttr
          if (call isUsedAttribute with sucAttr, (call data.getAntecedentAttributes)) == true then
            if (call isValidAttrSet with antCombIndex, sucCombIndex, antAttr, sucAttr) == true then
              set antecedent = call generateRecordComb with antCombIndex, antAttr, data
              set succedent = call generateRecordComb with sucCombIndex, sucAttr, data
              initialize element = new call associationRules constructor with antecedent, succedent,
                                                                          antAttr, sucAttr
              if result == null then
                set result = element
              else
                call element.setNext with result
                set result = element
              end if
            end if
          end if
        end if
    return result
end
```

Table 25. Pseudo-code for generate the attribute combination for the rule.

```
private static int[] generateAttrComb(int combIndex, int AttrNumComb, int attrNum,
                      int[] attrComb)
```

```
begin
  if AttrNumComb == 1 then
    initialize attrComb = new int[combIndex]
    for i = 0 to combIndex
      set attrComb[i] = 0
  else
    increment attrComb[0]
  end if
  for i = 0 to combIndex
    if attrComb[i] >= attrNum then
```

```
    set attrComb[i] = 0
    increment attrComb[i + 1]
   end if
  return attrComb
end
```

Table 26. Pseudo-code for validating if the attributes are in use.

```
private static boolean isUsedAttribute(int[] arrayEval, int[] arrayComp)

begin
  for i = 0 to arrayEval.length
    set member = false
    for index = 0 to  arrayComp.length
      if arrayEval[i] == arrayComp[index] then
        set member = true
        break
      end if
    if member != true then
      return false
    end if
  return true
end
```

Table 27. Pseudo-code for validating if the attributes are valid.

```
private static boolean isValidAttrSet(int antCombIndex, int sucCombIndex, int[] antAttr,
                                      int[] sucAttr)

begin

  // Antecedent attributes validation
  if (call validateAttributes with antCombIndex, antAttr) == true then

    // Succedent attributes validation
    if (call validateAttributes with sucCombIndex, sucAttr) == true then
      set valid = true
      initialize attrValidation = new int[antAttr.length + sucAttr.length]
      set attrValidation = call System.arraycopy with antAttr, 0, attrValidation, 0, antAttr.length
      set attrValidation = call System.arraycopy with sucAttr, 0, attrValidation, antAttr.length,
                                                                                  sucAttr.length
      call Arrays.sort with attrValidation

      // Validate that none of the attributes values are the same
      for i = 0 to (attrValidation.length – 1)
        if attrValidation[i] == attrValidation[i + 1] then
          set valid = false
          break
```

```
      end if
    return valid
   else
    return false
   end if
  else
   return false
  end if
end
```

Table 28. Pseudo-code for validating the attributes combination.

```
private static boolean validateAttributes(int combIndex, int[] attr)
```

```
begin
  set valid = true
  for i = 0 to combIndex
    for i2 = i to combIndex
      if attr[i] <  attr[i2] then
        set valid = false
        break
      end if
    if !valid then
      break
    end if
  return valid
end
```

Table 29. Pseudo-code for generate the records elements combination.

```
private static int[] generateRecordComb(int CombIndex, int[] attrComb, dataInput data)
```

```
begin
  initialize combination = new int[call data.getRecordsNumber]
  for i = 0 to CombIndex
    set values = call data.getRecordColumn with attrComb[i]

    // Add the records.
    for r = 0 to (call data.getRecordsNumber)
      set combination[r] = combination[r] + values[r]
  return combination
end
```

2.3.2.4. areaIdentification.java Class

Table 30. Pseudo-code for creating the differential matrix in the genetic algorithm.

public static String **consDifferenceMatrixQUARG**(dataInput data, associationRules rules)

```
begin
  set result = “”
  set selectedRulesCont = 0
  set rulesCont = 0

  // Cicle for each of the rules in the list.
  while rules != null
    set antMaxVal = call rules.getMaxAntecedentValue
    set sucMaxVal = call rules.getMaxSuccedentValue
    set actualCount = call fillActualCount with data, rules
    set expectationCount = call fillExpectationCount with data, actualCount
    set differentialCount = call fillDifferentialCount with expectationCount, actualCount
    set antAttrLength = call rules.getAntAttributesLength
    set sucAttrLength = call rules.getSucAttributesLength
    set squareTestVal = (antAttrLength + sucAttrLength) * 2
    set maxWidthAnt = call Math.max with (call Math.round with (antAttrLength * antMaxVal) / 5), 2
    set maxWidthSuc = call Math.max with (call Math.round with (sucAttrLength * sucMaxVal) / 5), 2

    // Genetic algorithm call
    set intArea = call identifyInterestArea with squareTestVal, differentialCount, antAttrLength

    // Area decomposition call
    set rule = call decomposeArea with intArea,
    set result += rule
    set rules = call rules.getNext
  endwhile
  set result += "Analized rules: " + rulesCont + ", selected rules: " + selectedRulesCont + ".\n"
  return result
end
```

Table 31. Pseudo-code for filling the actual count matrix.

private static int[][] **fillActualCount**(dataInput data, associationRules rules)

```
begin
  initialize matrixCount = new int[antMaxVal][sucMaxVal]
  for i = 0 to (call data.getRecordsNumber)
     set posX = (call rules.getAntecedentElement with i) - 1
     set posY = (call rules.getSuccedentElement with i) - 1
     increment matrixCount[posX][posY]
  return matrixCount
end
```

Table 32. Pseudo-code for filling the expectation count matrix.

```
private static float[][] fillExpectationCount(dataInput data, int[][] actualCount)
begin
  initialize expectationMatrix = new int[antMaxVal][sucMaxVal]
  for x = 0 to antMaxVal
     for y = 0 to sucMaxVal
       set columnSum = call sumColumnElements with actualCount, y, antMaxVal
       set rowSum = call sumRowElements with actualCount, x, sucMaxVal
       set recNum = call data.getRecordsNumber
       set expectationMatrix[x][y] = (columnSum * rowSum) / recNum
  return expectationMatrix
end
```

Table 33. Pseudo-code for filling the differential count matrix.

```
private static float[][] fillDifferentialCount(float[][] expectationCount, int[][] actualCount)
begin
  initialize differentialMatrix = new int[antMaxVal][sucMaxVal]
  for x = 0 to antMaxVal
     for y = 0 to sucMaxVal
       set differentialMatrix[x][y] = ((float) actualCount[x][y] – expectationCount[x][y]
  return  differentialMatrix
end
```

Table 34. Pseudo-code for identifying the interest areas for the genetic algorithm.

```
private static interestArea identifyInterestAreaQUARG(int squareTestVal,
                                                       float[][] differentialCount, int antComb)
begin
  set genNum = 21
  set chromeLength = 4
  set populationSize = (antMaxVal + sucMaxVal) * 2
  initialize population = new int[populationSize][chromeLength]
  initialize newPopulation = new int[populationSize][chromeLength]
  initialize finalArea = new int[antMaxVal][sucMaxVal]
  initialize population = new int[squareTestVal][chromeLength]
  initialize fitness = new float[populationSize]
  initialize indexFitness = new int[populationSize]
  set squaresNum = 0
  set population = call setPopulation with population, 0, populationSize
  for gen = 0 to genNum
    set indivNum = 0
    set fitness = call setFitness with fitness, population, populationSize, differentialCount, antComb
    set fitnessMaxVal = call getMaxValue with fitness
    for chrom = 0 to populationSize
      if fitness[chrom] > (fitnessMaxVal / 2) then
```

```
      set newPopulation[indivNum] = population[chrom]
      increment indivNum
    end if
  set newPopulation = call mutatePopulation with newPopulation, indivNum, chromeLength
  set population = call setPopulation with newPopulation, indivNum, populationSize
set indexFitness = call sortIndex with fitness, "descend"
for squares = 0 to squaresTestVal
  set overlapping = 0
  set squareSizeUpdate = 0
  initialize suppFinalArea = new int[antMaxVal][sucMaxVal]
  for x = population[indexFitness[squares]][0] to population[indexFitness[squares]][1]
    for y = population[indexFitness[squares]][2] to population[indexFitness[squares]][3]
      increment suppFinalArea[x - 1][y - 1]
  for x = 0 to antMaxVal
    for y = 0 to sucMaxVal
      if (finalArea[x][y] >= 1) and (suppFinalArea[x][y] >= 1 then
        increment overlapping
      end if
  set popAntDif = population[indexFitness[squares]][1] - population[indexFitness[squares]][0]
  set popAntDif = population[indexFitness[squares]][2] - population[indexFitness[squares]][3]
  set squareSizeUpdate = (call Math.abs with popAntDif) – (call Math.abs with popSucDif)
  if (squares == 0) or ((squares > 0) and (squareSizeUpdate > (0.5 * overlapping))) then
    set finalCoordinates[squaresNum][0] = population[indexFitness[squares]][0]
    set finalCoordinates[squaresNum][1] = population[indexFitness[squares]][1]
    set finalCoordinates[squaresNum][2] = population[indexFitness[squares]][2]
    set finalCoordinates[squaresNum][3] = population[indexFitness[squares]][3]
    set finalArea = call sumMatrices with finalArea, suppFinalArea, antMaxVal, sucMaxVal
    increment squaresNum
  end if
 return new call interestArea constructor with squaresNum, finalCoordinates
end
```

Table 35. Pseudo-code for setting the population matrix.

```
private static int[][] setPopulation(int[][] pMatrix, int pInitialPosition, int pSize)
```

```
begin
 for i = pInitialPosition to pSize
  set pMatrix[i][0] = call Math.ceil with ((call Math.random) * antMaxVal)
  set pMatrix[i][1] = (call Math.ceil with ((call Math.random) * (maxWidthAnt + 1))) + pMatrix[i][0]
  if pMatrix[i][1] > antMaxVal then
    set pMatrix[i][1] = antMaxVal
  end if
  set pMatrix[i][2] = call Math.ceil with ((call Math.random) * sucMaxVal)
  set pMatrix[i][3] = (call Math.ceil with ((call Math.random) * (maxWidthSuc + 1))) + pMatrix[i][2]
  if pMatrix[i][3] > sucMaxVal then
    set pMatrix[i][3] = sucMaxVal
```

```
  end if
 return pMatrix
end
```

Table 36. Pseudo-code for setting the fitness.

```
private static float[] setFitness(float[] fitness, int[][] population, int populationSize,
                                  float[][] differentialCount, int antComb)
begin
 for chrom = 0 to populationSize
  set antMinCoor = population[chrom][0]
  set antMaxCoor = population[chrom][1]
  set sucMinCoor = population[chrom][2]
  set sucMaxCoor = population[chrom][3]
  set sum = 0
  for y = antMinCoord to antMaxCoord
   for x = sucMinCoord to sucMaxCoord
    set sum = sum + differentialCount[y - 1][x - 1]
  if (antMaxCoord - antMinCoord) < antComb then
   set fitness[chrom] = -10000
  else
   set fitness[chrom] = sum
  end if
 return fitness
end
```

Table 37. Pseudo-code for the genetic mutation of the population.

```
private static int[][] mutatePopulation(int[][] newPopulation, int indivNum, int chromeLength)
begin
 set mutation = call Math.ceil with ((call Math.random) * indivNum)
 set mutationNum = call Math.ceil with ((call Math.random) * chromeLength)
 if (mutation >= indivNum) and (indivNum != 0) then
  set mutation = indivNum - 1
 end if
 if mutationNum == chromeLength then
  set mutationNum = chromeLength - 1
 end if
 if mutationNum == 0 then
  set newPopulation[mutation][mutationNum] = call Math.ceil with ((call Math.random) *
                                                                  antMaxVal)
  if newPopulation[mutation][mutationNum] > newPopulation[mutation][1] then
   set newPopulation[mutation][mutationNum] = newPopulation[mutation][1]
  end if
 else if mutationNum == 1 then
  set newPopulation[mutation][mutationNum] = call Math.ceil with ((call Math.random) *
```

```
                                                                    antMaxVal)
    if newPopulation[mutation][mutationNum] < newPopulation[mutation][0] then
      set newPopulation[mutation][mutationNum] = newPopulation[mutation][0]
    end if
  else if mutationNum == 2 then
    set newPopulation[mutation][mutationNum] = call Math.ceil with ((call Math.random) *
                                                                    sucMaxVal)
    if newPopulation[mutation][mutationNum] > newPopulation[mutation][3] then
      set newPopulation[mutation][mutationNum] = newPopulation[mutation][3]
    end if
  else if mutationNum == 3 then
    set newPopulation[mutation][mutationNum] = call Math.ceil with ((call Math.random) *
                                                                    sucMaxVal)
    if newPopulation[mutation][mutationNum] < newPopulation[mutation][2] then
      set newPopulation[mutation][mutationNum] = newPopulation[mutation][2]
    end if
  end if
  return newPopulation
end
```

Table 38. Pseudo-code for for creating the differential matrix in the full search algorithm.

```
public static String consDifferenceMatrixFull(dataInput data, associationRules rules)
begin
  set resultBuffer = new StringBuffer with 10000
  while rules != null
    set antRecIndex = call generateRecordsIndexation with (call getAntecedent)
    set sucRecIndex = call generateRecordsIndexation with (call getSuccedent)
    set antMaxVal = antRecIndex.length
    set sucMaxVal = sucRecIndex.length
    set actualCount = call fillActualCountAlt with data, rules, antRecIndex, sucRecIndex
    set expectationCount = call fillExpectationCount with data, actualCount
    set differentialCount = call fillDifferentialCount with expectationCount, actualCount
    set intArea = call identifyInterestAreaFull with differentialCount, antRecIndex, sucRecIndex
    set rule = call decomposeArea with intArea, data, rules
    call resultBuffer.append with rule
    set rules = call rules.getNext
  return (call resultBuffer.toString)
end
```

Table 39. Pseudo-code for generate an array of indexes for the records.

```
private static int[] generateRecordsIndexation(int[] records)
begin
  initialize indexation = new int[1]
  set recordsCopy = call Arrays.copyOf with records, recordsLength
```

```
  call Arrays.sort with recordsCopy
  set indexation[0] = recordsCopy[0]
  for i = 1 to records.length
    if recordsCopy[i] != indexation[index.length – 1] then
      initialize tempIndexation = new int[index.length + 1]
      call System.arraycopy with indexation, 0, tempIndexation, 0, index.length
      set tempIndexation[index.length] = recordsCopy[i]
      set indexation = tempIndexation
    end if
  return indexation
end
```

Table 40. Pseudo-code for filling the actual count matrix for the full search method.

```
private static int[][] fillActualCountAlt(dataInput data, associationRules rules, int[] antRecordIndex,
                                          int[] sucRecordIndex)
begin
  set dataRecNum = call data.getRecordsNumber
  initialize matrixCount = new int[antMaxVal][sucMaxVal]
  for i = 0 to dataRecNum
    set posX = call Arrays.binarySearch with antRecordIndex, (call rules.getAntecedentElement with i)
    set posX = call Arrays.binarySearch with sucRecordIndex, (call rules.getSuccedentElement with i)
    increment matrixCount[posX][posY]
  return matrixCount
end
```

Table 41. Pseudo-code for identifying the interest areas for the full search algorithm.

```
private static interestArea identifyInterestAreaFull(float[][] differentialCount, int[] antRecIndex,
                                                     int[] sucRecIndex)
begin
  set coordinates = null
  set values = null
  for posIniX = 0 to antMaxVal
    for posIniY = 0 to sucMaxVal
      for posFinX = posIniX to antMaxVal
        for posFinY = posIniY to sucMaxVal
          set value = call getCoordinatesValue with differentialCount, posIniX, posFinX, posIniY,
                                                    posFinY
          set coordinate = {posIniX, posFinX, posIniY, posFinY}
          if value > 0 then
            if coordinates == null then
              initialize coordinates = new int[1][4]
              set coordinates[0] = coordinate
              initialize values = new float[1]
            end if
```

```
        else
          set overlapping = call findOverlap with value, coordinate
          if !overlapping then
            call addCoordinates with coordinate, value
          end if
        end if
  if coordinates != null then
    set finalCoordinates = call getRealCoordinates with antRecIndex, sucRecIndex
    return (call interestArea constructor with finalCoordinates.length, finalCoordinates)
  else
    return (call interestArea constructor with 0, null)
  end if
end
```

Table 42. Pseudo-code for getting the value of the sum if elements in the differential count matrix.

```
private static float getCoordinatesValue(float[][] differentialCount, int posIniX, int posFinX,
                                         int posIniY, int posFinY)
begin
  set value = 0
  for index = posIniX to posFinX
    set row = call Arrays.copyOfRange with differentialCount[index], posIniY, (posFinY + 1)
    set value = value + (call sumArrayElements with row)
  return value
end
```

Table 43. Pseudo-code for finding overlap of the elements in the interest areas.

```
private static boolean findOverlap(float value, int[] coordinate)
begin
  set overlap = false
  set valid = true
  for i = 0 to coordinates.length
    set actualCoordinate = coordinates[i]
    set actualValue = values[i]
    if ((coordinate[0] >= actualCoordinate[0] and coordinate[0] <= actualCoordinate[1]) or
       (coordinate[1] >= actualCoordinate[0] and coordinate[1] <= actualCoordinate[1])) and
       ((coordinate[2] >= actualCoordinate[2] and coordinate[2] <= actualCoordinate[3]) or
       (coordinate[3] >= actualCoordinate[2] and coordinate[3] <= actualCoordinate[3])) then
      set overlap  = true
      set overlapArea = call getOverlapArea with coordinate, actualCoordinate
      set area = ((call Math.abs with (coordinate[0] – coordinate[1])) + 1) *
                 ((call Math.abs with (coordinate[2] – coordinate[3])) + 1)
      set areaDiff = overlapArea / area
      set valueDiff = value / actualValue
      if areaDiff > valueDiff then
```

```
      set valid = false
      break
    end if
  end if
 if overlap and valid then
  call addCoordinates with coordinate, value
 end if
 return overlap
end
```

Table 44. Pseudo-code for determine how many elements conforms the overlap area.

```
private static float getOverlapArea(int[] coordinate, int[] actualCoordinate)
begin
 set minValX = call Math.min with coordinate[0], actualCoordinate[0]
 set maxValX = call Math.max with coordinate[1], actualCoordinate[1]
 set minValY = call Math.min with coordinate[2], actualCoordinate[2]
 set maxValY = call Math.max with coordinate[3], actualCoordinate[3]
 set valX = 0
 set valY = 0
 for x = minValX to maxValX
  if (x >= actualCoordinate[0] and x <= actualCoordinate[1]) and
     (x >= coordinate[0] and x <= coordinate[1]) then
   increment valX
  end if
 for y = minValY to maxValY
  if (y >= actualCoordinate[2] and y <= actualCoordinate[3]) and
     (y >= coordinate[2] and y <= coordinate[3]) then
   increment valY
  end if
 return (valX * valY)
end
```

Table 45. Pseudo-code for adding a coordinate to the list of accepted ones.

```
private static void addCoordinates(int[] coordinate, float value)
begin
 initialize tempCoordinates = new int[coordinates.length + 1][4]
 call System.arraycopy with coordinates, 0, tempCoordinates, 0, coordinates.length
 set tempCoordinates[coordinates.length] = coordinate
 set coordinates = tempCoordinates
 initialize tempValues = new float[values.length + 1]
 call System.arraycopy with values, 0, tempValues, 0, values.length
 set tempValues[values.length] = value
 set values = tempValues
end
```

Table 46. Pseudo-code for changing the indexed positions for the real ones.

private static int[][] **getRealCoordinates**(int[] antRecIndex, int[] sucRecIndex)

begin
  initialize *finalCoordinates* = **new** int[*coordinates*.length][4]
  for *x* = 0 **to** *coordinates*.length
  set *coordinate* = {*antRecIndex*[*coordinates*[*x*][0], *antRecIndex*[*coordinates*[*x*][1],
  sucRecIndex[*coordinates*[*x*][2], *sucRecIndex*[*coordinates*[*x*][3]}
  set *finalCoordinates*[*x*] = *coordinate*
  return *finalCoordinates*
end

Table 47. Pseudo-code for creating the general rule compound result.

private static String **generalRuleResult**(dataInput data, associationRules rules)

begin
  set *rulesAntAttr* = **call** *rules*.getAntAttributes
  set *antAttrLength* = **call** *rules*.getAntAttributesLength
  set *rulesSucAttr* = **call** *rules*.getSucAttributes
  set *sucAttrLength* = **call** *rules*.getSucAttributesLength
  set *ruleData* = **new** StringBuilder with 50
  for *index* = 0 **to** *antAttrLength*
  set *attrName* = **call** *data*.getAttributesElement with *rulesAntAttr*[*index*]
  call *ruleData*.append with *attrName*
  for *index* = 0 **to** *sucAttrLength*
  set *attrName* = **call** *data*.getAttributesElement with *rulesSucAttr*[*index*]
  call *ruleData*.append with *attrName*
  return (**call** *ruleData*.toString)
end

2.3.2.5. areaDecomposition.java Class

Table 48. Pseudo-code for decomposing the selected interest areas.

public static String **decomposeArea**(interestArea intArea, dataInput data, associationRules rules)

begin
  set *result* = “”
  set *antAttrNum* = **call** *rules*.getAntAttributesLength
  set *sucAttrNum* = **call** *rules*.getSucAttributesLength
  for *cont* = 0 **to** (**call** *intArea*.getSquaresNum)
  initialize *decArea* = **new call** decomposedArea constructor
 	
  // Antecedent decomposition.
  if (**call** *rules*.getAntAttributesLength) == 1 **then**
  call *decArea*.incrementAntNumber
  initialize *tempMatrix* = **new** int[1][2]

```
    set tempMatrix[0][0] = call Math.min with (call intArea.getCoordinatesElement with cont, 0),
                                              (call  intArea.getCoordinatesElement with cont, 1)
    set tempMatrix[0][1] = call Math.max with (call intArea.getCoordinatesElement with cont, 0),
                                              (call  intArea.getCoordinatesElement with cont, 1)
    call decArea.setDecompAntecedent with tempMatrix
   else
    call startAntDecomposition with (new int[antAttrNum), 1,
                                     (call intArea.getCoordinatesElement with cont, 0),
                                     (call intArea.getCoordinatesElement with cont, 1)
    if (call decArea.getDecompAntNumber) == 0 then
     initialize tempMatrix = new int[1][antAttrNum * 2]
     call Arrays.fill with tempMatrix[0], 1
     call decArea.setDecompAntecedent with tempMatrix
     call decArea.incrementAntNumber
    end if
   end if

   // Succedent decomposition.
   if (call rules.getSucAttributesLength) == 1 then
    call decArea.incrementSucNumber
    initialize tempMatrix = new int[1][2]
    set tempMatrix[0][0] = call Math.min with (call intArea.getCoordinatesElements with cont, 2),
                                              (call intArea.getCoordinatesElements with cont, 3)
    set tempMatrix[0][1] = call Math.max with (call intArea.getCoordinatesElements with cont, 2),
                                              (call intArea.getCoordinatesElements with cont, 3)
    call decArea.setDecompSuccedent with tempMatrix
   else
    call startSucDecomposition with (new int[sucAttrNum]), 1,
                                     (call intArea.getCoordinatesElement with cont, 2),
                                     (call intArea.getCoordinatesElement with cont, 3)
    if (call decArea.getDecompSucNumber) == 0 then
     initialize tempMatrix = new int[1][sucAttrNum * 2]
     call Arrays.fill with tempMatrix[0], 1
     call decArea.setDecompSuccedent with tempMatrix
     call decArea.incrementSucNumber
    end if
   end if
   // Rule selection.
   set rule = call selectRule with decArea, data, rules
   set result = result + (call rule.getRuleResult)
   set areaIdentification.rulesCount =  areaIdentification.rulesCount
                                     + (call rule.getVerificationsNumber)
   set areaIdentification.selectedRulesCount =  areaIdentification.selectedRulesCount
                                             + (call rule.getRulesNumber)
  return result
end
```

Table 49. Pseudo-code for starting the recursive method for the antecedent decomposition.

```
private static void startAntDecomposition(int[] combArray,int cyclesCount, int initialAnt,
                                          int finalAnt)
begin
  for start = 1 to areaIdentification.antMaxVal
    if cyclesCount == antAttrNum then
      set combArray[cyclesCount - 1] = start
      if (call sumArrayElements with combArray) == initialAnt then
        call stopAntDecomposition with combArray, (new int[antAttrNum]), 1, finalAnt
      end if
    else
      set combArray[cyclesCount - 1] = start
      call startAntDecomposition with combArray, (cyclesCount + 1), initialAnt, finalAnt
    end if
end
```

Table 50. Pseudo-code for stopping the recursive method for the antecedent decomposition.

```
private static void stopAntDecomposition(int[] combArrayInit, int[] combArrayFin, int cyclesCount,
                                         int finalAnt)
begin
  set limit = call Math.min with (combArrayInit[cyclesCount - 1] + 1), areaIdentification.antMaxVal
  for stop = combArray[cyclesCount - 1] to limit
    if cyclesCount == antAttrNum then
      set combArrayFin[cyclesCount - 1] = stop
      if (call sumArrayElements with combArrayFin) == finalAnt then
        initialize tempArray = new int[antAttrNum * 2]
        call System.arraycopy with combArrayInit, 0, tempArray, 0, antAttrNum
        call System.arraycopy with combArrayFin, 0, tempArray, antAttrNum, antAttrNum
        set tempMatrix = call modifyDecompMatrix with (call decArea.getDecompAntecedent),
                                                      (call decArea.getDecompAntNumber),
                                                      tempArray
        call decArea.setDecompAntecedent with tempMatrix
        call decArea.incrementAntNumber
      end if
    else
      set combArrayFin[cyclesCount - 1] = stop
      call stopAntDecomposition with combArrayInit, combArrayFin, (cyclesCount +1), finalAnt
    end if
end
```

Table 51. Pseudo-code for starting the recursive method for the succedent decomposition.

```
private static void startSucDecomposition(int[] combArray,int cyclesCount, int initialSuc,
                                          int finalSuc)
begin
  for start = 1 to areaIdentification.sucMaxVal
    if cyclesCount == sucAttrNum then
      set combArray[cyclesCount - 1] = start
      if (call sumArrayElements with combArray) == initialSuc then
        call stopSucDecomposition with combArray, (new int[sucAttrNum]), 1, finalSuc
      end if
    else
      set combArray[cyclesCount - 1] = start
      call startSucDecomposition with combArray, (cyclesCount + 1), initialSuc, finalSuc
    end if
end
```

Table 52. Pseudo-code for stopping the recursive method for the succedent decomposition.

```
private static void stopSucDecomposition(int[] combArrayInit, int[] combArrayFin, int cyclesCount,
                                         int finalSuc)
begin
  set limit = call Math.min with (combArrayInit[cyclesCount - 1] + 1), areaIdentification.sucMaxVal
  for stop = combArray[cyclesCount - 1] to limit
    if cyclesCount == sucAttrNum then
      set combArrayFin[cyclesCount - 1] = stop
      if (call sumArrayElements with combArrayFin) == finalSuc then
        initialize tempArray = new int[sucAttrNum * 2]
        call System.arraycopy with combArrayInit, 0, tempArray, 0, sucAttrNum
        call System.arraycopy with combArrayFin, 0, tempArray, sucAttrNum, sucAttrNum
        set tempMatrix = call modifyDecompMatrix with (call decArea.getDecompSuccedent),
                                                      (call decArea.getDecompSucNumber),
                                                      tempArray
        call decArea.setDecompSuccecedent with tempMatrix
        call decArea.incrementSucNumber
      end if
    else
      set combArrayFin[cyclesCount - 1] = stop
      call stopSucDecomposition with combArrayInit, combArrayFin, (cyclesCount +1), finalSuc
    end if
end
```

Table 53. Pseudo-code for modifying the decomposition result matrix.

```
private static int[][] modifyDecompMatrix(int[][] decMatrix, int decIndex, int[] tempArray)
```

```
begin
  initialize tempMatrix = new int[decIndex + 1][tempArray.length]
  if decMatrix == null then
    set tempMatrix[decIndex] = tempArray
  else
    call System.arraycopy with decMatrix, 0, tempMatrix, 0, decIndex
    set tempMatrix[decIndex] = tempArray
  end if
  return tempMatrix
end
```

2.3.2.6. ruleSelection.java

Table 54. Pseudo-code for selecting rules as result.

```
public static analyzedRule selectRule(decomposedArea decArea, dataInput data,
                                       associationRules rules)
```

```
begin
  initialize result = new call analysedRule constructor
  set validRule = false
  set initialVal = 0
  set finalVal = 0
  set resultConf = ""
  set resultSupp = ""
  set resultLift = ""
  set antAttrLength = call rules.getAntAttributesLength
  set sucAttrLength = call rules.getSucAttributesLength
  for i = 0 to (call decArea.getDecompAntNumber)
    for j = 0 to (call decArea.getDecompSucNumber)
      call result.incrementVerificationsNumber
      set AntCompSucComp = 0
      set AntUncompSucComp = 0
      set AntCompSucUncomp = 0
      for record = 0 to (call data.getRecordsNumber)

        // Antecedent testing.
        set antTest = call testAttributes with antAttrLength, (call data.getRecords), record,
                                              (call rules.getAntAttributes),
                                              (call decArea.getDecompAntecedent), i

        // Succedent testing.
        set sucTest = call testAttributes with sucAttrLength, (call data.getRecords), record,
                                              (call rules.getSucAttributes),
                                              (call decArea.getDecompSuccedent), j
```

```
      if (antTest == true) and (sucTest == true) then
        increment AntCompSucComp
      else if (antTest == true) and (sucTest == false) then
        increment AntCompSucUncomp
      else if (antTest == false) and (sucTest == true) then
        increment AntUncompSucComp
      end if
    set support = call getRuleSupport with AntCompSucComp, (call data.getRecordsNumber)
    set confidence = call getRuleConfidence with AntCompSucComp, AntCompSucUncomp
    set lift = call getRuleLift with AntCompSucComp, AntCompSucUncomp, AntUncompSucComp,
                                        (call data.getRecordsNumber)
    if (confidence > (call data.getConf)) and (support > (call data.getSupp))
                and (lift > (call data.getLift) then
      set validRule = true
      set initialVal = i
      set finalVal = j
      set resultConf = call shortenValue with confidence
      set resultSupp = call shortenValue with support
      set resultLift = call shortenValue with lift
    end if
  if validRule == true then
    call result.incrementRulesNumber
    set ruleData = ""

    // Antecedent result creation.
    set ruleData = ruleData + (call createResult with antAttrLength, data, (call
rules.getAntAttributes),                                                    (call
decArea.getDecompAntecedent), initalVal)

    // Succedent result creation.
    set ruleData = ruleData + (call createResult with sucAttrLength, data, (call
rules.getSucAttributes),                                                    (call
decArea.getDecompSuccedent),
          finalVal)
    set ruleData = ruleData + "conf = " + resultConf + " supp = " + resultSupp  + " lift = " + resultLift
                            + "\n"
    call result.addRule with ruleData
  end if
  return result
end
```

Table 55. Pseudo-code for testing the attributes values.

```
private static boolean testAttributes(int attrLength, int[][] records, int recordPos, int[] attrList,
                                      int[][] decompArea, int index)
begin
  set testNumber = 0
  set validAttr = false
  for count = 1 to attrLength
    set recordValue = records[recordPos][attrList[count + 1]]
    set antValueIni = decompArea[index][2 * (count - 1)]
    set antValueFin = decompArea[index][(2 * (count - 1)) + 1]
    if (recordValue >= antValueIni) and (recordValue <= antValueFin) then
      increment testNumber
    end if
  if testNumber == attrLength then
    set validAttr = true
  end if
  return validAttr
end
```

Table 56. Pseudo-code for creating the result output.

```
private static String createResult(int attrLength, dataInput data, int[] attrList, int[][] decompArea,
                                   int posVal)
begin
  set ruleData = ""
  for index = 1 to attrLength
    set attrName = call data.getAttributesElement with attrList[index - 1]
    set initialPos = call Math.ceil with decompArea[posVal][2 * (index - 1)]
    set finalPos = call Math.floor with decompArea[posVal][(2 * (index - 1)) + 1]
    if initialPos != finalPos then
      set ruleData = ruleData + attrName + " = " + initialPos + ".." + finalPos + " "
    else
      set ruleData = ruleData + attrName + " = " + initialPos + " "
    end if
  return ruleData
end
```

Table 57. Pseudo-code for calculating the current rule support value.

```
private static float getRuleSupport(int AntCompSucComp, int recordsNumber)
begin
  set support = 0
  set records = (float) recordsNumber
  set attrComp = (float) AntCompSucComp
  if records != 0 then
    set support = attrComp / records
```

```
  end if
  return support
end
```

Table 58. Pseudo-code for calculating the current rule confidence value.

```
private static float getRuleConfidence(int AntCompSucComp, int AntCompSucUncomp)
begin
  set confidence = 0
  set antCsucC = (float) AntCompSucComp
  set antCsucU = (float) AntCompSucUncomp
  if (antCsucC + antCsucU) != 0 then
    set confidence = antCsucC / ( antCsucC + antCsucU)
  end if
  return confidence
end
```

Table 59. Pseudo-code for calculating the current rule lift value.

```
private static float getRuleLift(int AntCompSucComp, int AntCompSucUncomp,
                                 int AntUncompSucComp, int recordsNumber)
begin
  set lift = 0
  set records = (float) recordsNumber
  set antCsucC = (float) AntCompSucComp
  set antCsucU = (float) AntCompSucUncomp
  set antUsucC = (float) AntUncompSucComp
  if ((antCsucC + antCsucU) != 0) and (records != 0) then
    set lift = (antCsucC / (antCsucC + antCsucU))  / ((antCsucC + antUsucC) / records)
  end if
  return lift
end
```

Table 60. Pseudo-code for shortening the float values of support, confidence and lift.

```
private static String shortenValue(float number)
begin
  if (call String.valueOf with number).length > 4 then
    return (call (call String.valueOf with number).substring with 0, 4)
  else
    return (call String.valueOf with number)
  end if
end
```

2.3.3. quargObjects Package

2.3.3.1. dataInput.java Class

Table 61. Pseudo-code for the data input object constructor.

public **dataInput**(String attr, String rec, int antMin, int antMax,int sucMin, int sucMax, String antAttr, String sucAttr, float c, float l, float s)

begin
  set *attributes* = *attr*.split(“;”)
  set *recordsRows* = *rec*.split(“,”)
  initialize *records* = **new** int[*recordsRows*.length][*attributes*.length]
  for *y* = 0 **to** *recordsRows*.length
  set *rowData* = *recordsRows*[*y*].split(“;”)
  for *x* = 0 **to** *rowData*.length
  set *records*[*y*][*x*] = integer value of *rowData*[*x*]
  set *attributesNumber* = *attributes*.length
  set *recordsNumber* = *recordsRows*.length
  set *antecedentMin* = *antMin*
  set *antecedentMax* = *antMax*
  set *succedentMin* = *sucMin*
  set *succedentMax* = *sucMax*
  set antValues = antAttr.split(“;”)
  initialize *antecedentAttributes* = **new** int[*antValues*.length]
  for *i* = 0 **to** *antValues*.length
  set *antecedentAttributes*[i] = integer value of *antValues*[i]
  set *sucValues* = *sucAttr*.split(“;”)
  initialize *succedentAttributes* = **new** int[*sucValues*.length]
  for *i* = 0 **to** *sucValues*.length
  set *succedentAttributes*[i] = integer value of *sucValues*[i]
  set *conf* = *c*
  set *lift* = *l*
  set *supp* = *s*
end

Table 62. Pseudo-code for the attributes item selection.

public String **getAttributesElement**(int index)
begin   **return** *attributes*[*index*] **end**

Table 63. Pseudo-code for the records matrix selection.

public int[][] **getRecords**()
begin   **return** *records*

end

Table 64. Pseudo-code for the antecedent minimum combination value selection.

public int **getAntecedentMin**()
begin   **return** *antecedentMin* **end**

Table 65. Pseudo-code for the antecedent maximum combination value selection.

public int **getAntecedentMax**()
begin   **return** *antecedentMax* **end**

Table 66. Pseudo-code for the succedent minimum combination value selection.

public int **getSuccedentMin**()
begin   **return** suc*cedentMin* **end**

Table 67. Pseudo-code for the succedent maximum combination value selection.

public int **getSuccedentMax**()
begin   **return** suc*cedentMax* **end**

Table 68. Pseudo-code for the antecedent attributes selection.

public int[] **getAntecedentAttributes**()
begin   **return** *antecedentAttributes* **end**

Table 69. Pseudo-code for the succedent attributes selection.

public int[] **getSuccedentAttributes**()
begin   **return** suc*cedentAttributes* **end**

Table 70. Pseudo-code for the confidence value selection.

public float **getConf**()
begin   **return** *conf* **end**

Table 71. Pseudo-code for the lift value selection.

public float **getLift**()
begin   **return** *lift* **end**

Table 72. Pseudo-code for the support value selection.

public float **getSupp**()
begin   **return** *supp* **end**

Table 73. Pseudo-code for the total attributes number selection.

public int **getAttributesNumber**()
begin   **return** *attributesNumber* **end**

Table 74. Pseudo-code for the total records number selection.

public int **getRecordsNumber**()
begin   **return** records*Number* **end**

Table 75. Pseudo-code for the records specific column selection.

public int[] **getRecordColumn**(int index)
begin   **return** the **call** getColumn with *records*, *index*, *recordsNumber* **end**

2.3.3.2. associationRules.java Class

Table 76. Pseudo-code for the association rule object constructor.

public **associationRules**(int[] ant, int[] suc, int[] antCols, int[] sucCols)

begin
  set *antecedent* = *ant*
  set *succedent* = *suc*
  set *antAttributes* = **call** Arrays.copyOf with *antCols*, *antCols*.length
  set *sucAttributes* = **call** Arrays.copyOf with *sucCols*, *sucCols*.length
  set *next* = null
end

Table 77. Pseudo-code for the maximum antecedent value selection.

public int **getMaxAntecedentValue**()
begin   **return** the **call** getMaxValue with *antecedent* **end**

Table 78. Pseudo-code for the maximum succedent value selection.

public int **getMaxSuccedentValue**()
begin   **return** the **call** getMaxValue with suc*cedent* **end**

Table 79. Pseudo-code for the antecedent element selection.

public int **getAntecedentElement**(int index)
begin   **return** *antecedent*[*index*] **end**

Table 80. Pseudo-code for the succedent element selection.

public int **getSuccedentElement**(int index)
begin   **return** suc*cedent*[*index*] **end**

Table 81. Pseudo-code for the next rule selection.

public associationRules **getNext**()
begin   **return** *next* **end**

Table 82. Pseudo-code for obtaining the antecedent attributes length.

public int **getAntAttributesLength**()
begin   **return** *antAttributes*.length **end**

Table 83. Pseudo-code for obtaining the succedent attributes length.

public int **getSucAttributesLength**()
begin   **return** suc*Attributes*.length **end**

Table 84. Pseudo-code for getting the array of antecedents indexes.

public int[] **getAntAttributes**()
begin   **return** *antAttributes* **end**

Table 85. Pseudo-code for getting the array of succedents indexes.

public int[] **getSucAttributes**()
begin   **return** *sucAttributes* **end**

Table 86. Pseudo-code for the next rule list element set up.

public void **setNext**(associationRules n)
begin   **set** *next = n* **end**

2.3.3.3. interestArea.java Class

Table 87. Pseudo-code for the interest area object constructor.

public **interestArea**(int sNum, int[][] coord)

begin
  set *squaresNum = sNum*
  set *coordinates = coord*
end

Table 88. Pseudo-code for getting the squares number value.

public int **getSquaresNum**()
begin   **return** *squaresNum* **end**

Table 89. Pseudo-code for getting the specific coordinate value.

public int **getCoordinatesElement**(int indexA, int indexB)
begin   **return** *coordinates*[*indexA*][*indexB*] **end**

2.3.3.4. decomposedArea.java Class

Table 90. Pseudo-code for the decomposed area object constructor.

public **decomposedArea**()

begin
  set *decompAntecedent* = null
  set *decompSuccedent* = null
  set *decompAntNumber* = 0
  set *decompSucNumber* = 0
end

Table 91. Pseudo-code for the decomposed antecedent matrix.

public int[][] **getDecompAntecedent**()
begin   **return** *decompAntecedent* **end**

Table 92. Pseudo-code for the decomposed succedent matrix.

public int **getDecompSuccedent**()
begin   **return** *decompSuccedent* **end**

Table 93. Pseudo-code for the decomposed antecedent number selection.

public int **getDecompAntNumber**()
begin   **return** *decompAntNumber* **end**

Table 94. Pseudo-code for the decomposed succedent number selection.

public int **getDecompSucNumber**()
begin   **return** *decompSucNumber* **end**

Table 95. Pseudo-code for incrementing the antecedent number.

public void **incrementAntNumber**()
begin   **increment** *decompAntNumber* **end**

Table 96. Pseudo-code for incrementing the succedent number.

public void **incrementSucNumber**()
begin   **increment** *decompSucNumber* **end**

Table 97. Pseudo-code for setting up the decomposed antecedent matrix.

public void **setDecompAntecedent**(int[][] value)
begin   **set** *decompAntecedent = value* **end**

Table 98. Pseudo-code for setting up the decomposed succedent matrix.

public void **setDecompSuccedent**(int[][] value)
begin   **set** *decompSuccedent = value* **end**

2.3.3.5. analyzedRule.java Class

Table 99. Pseudo-code for the analyzed rule object constructor.

public **analyzedRule**()

begin
  set *ruleResult* = “”
  set *rulesNumber* = 0
  set *verificationsNumber* = 0
end

Table 100. Pseudo-code for getting the rule result value.

public String **getRuleResult**()
begin   **return** *ruleResult* **end**

Table 101. Pseudo-code for selecting the rules result number.

public int **getRulesNumber**()
begin   **return** *rulesNumber* **end**

Table 102. Pseudo-code for selecting the rules verifications number.

public int **getVerificationsNumber**()
begin   **return** *verificationsNumber* **end**

Table 103. Pseudo-code for incrementing the rules result number.

public void **incrementRulesNumber**()
begin   **increment** *rulesNumber* **end**

Table 104. Pseudo-code for incrementing the rules verifications number.

public void **incrementVerificationsNumber**()
begin   **increment** *verificationsNumber* **end**

Table 105. Pseudo-code for adding a rule to the result.

public void **addRule**(String rule)
begin   **set** *ruleResult = ruleResult + rule* **end**

2.4. Source Code

2.4.1. commonFunctions Package

2.4.1.1. arrays.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package commonFunctions;

import java.util.Arrays;

/**
 * Handle the functions related with the arrays in the application.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class arrays {

   /**
    * Search the element with the maximum integer value from an array.
    * @param array the array to analyze.
    * @return the element with the maximum value.
    */
   public static int getMaxValue(int[] array) {
      int max = 0;
      final int length = array.length;

      for (int i = 0; i < length; i++) {
         if (array[i] > max) {
            max = array[i];
         }
      }

      return max;
   }

   /**
    * Search the element with the maximum float value from an array.
    * @param array the array to analyze.
    * @return the element with the maximum value.
    */
```

```
public static float getMaxValue(float[] array) {
    float max = 0;
    final int length = array.length;

    for (int i = 0; i < length; i++){
        if(array[i] > max) {
            max = array[i];
        }
    }

    return max;
}

/**
 * Search the element with the minimum float value from an array.
 * @param array the array to analyze.
 * @return the element with the minimum value.
 */
public static float getMinValue(float[] array) {
    float min = array[0];
    final int length = array.length;

    for (int i = 1; i < length; i++){
        if(array[i] < min) {
            min = array[i];
        }
    }

    return min;
}

/**
 * Generate an sorted array with the index of the elements of the
 * input array.
 * @param array the array to evaluate.
 * @param typeSort the way to sort the array (ascend/descend).
 * @return an sorted array with the index.
 */
public static int[] sortIndex(float[] array, String typeSort) {
    final int length = array.length;
    float[] arrayCopy = Arrays.copyOf(array, length);
    float[] arraySorted = Arrays.copyOf(array, length);
    int[] index = new int[length];

    Arrays.sort(arraySorted);

    if (typeSort.equals("descend")) {
```

```
        arraySorted = reverseArray(arraySorted);
    }

    for (int i = 0; i < length; i++) {
        int pos = 0;

        for (int j = 0; j < length; j++) {
            if ((arrayCopy[j] != -50000)
                    && (arraySorted[i] == arrayCopy[j])) {
                arrayCopy[j] = -50000;
                break;
            } else {
                pos++;
            }
        }
        index[i] = pos;
    }

    return index;
}

/**
 * Reverse the order of the elements in the array.
 * @param array the array to be reversed.
 * @return the reverse array.
 */
public static float[] reverseArray(float[] array) {
    final int length = array.length;
    float[] reversed = new float[length];

    for (int i = 0; i < length; i++) {
        reversed[i] = array[length - i - 1];
    }

    return reversed;
}

/**
 * Sum the integer elements in an array.
 * @param array the array to be evaluate.
 * @return the sum integer value of the elements.
 */
public static int sumArrayElements(int[] array){
    int result = 0;
    final int length = array.length;

    for (int index = 0; index < length; index++) {
```

```
            result += array[index];
        }

        return result;
    }

    /**
     * Sum the float elements in an array.
     * @param array the array to be evaluate.
     * @return the sum float value of the elements.
     */
    public static float sumArrayElements(float[] array){
        float result = 0;
        final int length = array.length;

        for (int index = 0; index < length; index++) {
            result += array[index];
        }

        return result;
    }
}
```

2.4.1.2. math.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package commonFunctions;

/**
 * Handle the functions related with the numeric functionalities in the
 * application.
 *
 * @version 1.0 Jan 2012
 * @author Adalberto Cubillo
 */
public final class math {

    /**
     * Get the value of the power, is similar that apply Math.pow().
     * @param base the multiplying value.
     * @param expt the number of times that the base is going to be multiply.
     * @return the power.
```

```
     */
    public static int getPower(int base, int expt) {
        int result = 1;

        for (int i = 0; i < expt; i++) {
                result *= base;
        }
        return result;
    }
}
```

2.4.1.3. matrices.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package commonFunctions;

/**
 * Handle the functions related with the matrices in the application.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class matrices {

    /**
     * Generate an array with a certain column from an integer matrix.
     * @param matrix the matrix where the column will be extract.
     * @param columnIndex the position of the column in the matrix.
     * @param columnLength the size of the column.
     * @return an array with the elements of the column.
     */
    public static int[] getColumn(int[][] matrix, int columnIndex,
            int columnLength) {
        int[] column = new int[columnLength];

        for (int i = 0; i < columnLength; i++) {
            column[i] = matrix[i][columnIndex];
        }

        return column;
    }
```

```
/**
 * Generate an array with a certain column from an float matrix.
 * @param matrix the matrix where the column will be extract.
 * @param columnIndex the position of the column in the matrix.
 * @param columnLength the size of the column.
 * @return an array with the elements of the column.
 */
public static float[] getColumn(float[][] matrix, int columnIndex,
        int columnLength) {
    float[] column = new float[columnLength];

    for (int i = 0; i < columnLength; i++) {
        column[i] = matrix[i][columnIndex];
    }

    return column;
}

/**
 * Get the sum of the integer elements of a certain row.
 * @param matrix the matrix to read.
 * @param row the row in which make the sum.
 * @param columns quantity of columns in the matrix.
 * @return the sum of the column elements.
 */
public static int sumRowElements(int[][] matrix, int row, int columns) {
    int result = 0;

    for (int i = 0; i < columns; i++) {
        result += matrix[row][i];
    }

    return result;
}

/**
 * Get the sum of the integer elements of a certain column.
 * @param matrix the matrix to read.
 * @param column the column in which make the sum.
 * @param rows quantity of rows in the matrix.
 * @return the sum of the column elements.
 */
public static int sumColumnElements(int[][] matrix, int column, int rows) {
    int result = 0;

    for (int i = 0; i < rows; i++) {
        result += matrix[i][column];
```

```
    }

    return result;
}

/**
 * Add the elements of two integer matrices of the same size.
 * @param matrix1 the first matrix.
 * @param matrix2 the second matrix.
 * @param rowLength matrices row quantity.
 * @param columnLength matrices column quantity.
 * @return a matrix with the sum of both input matrices.
 */
public static int[][] sumMatrices(int[][] matrix1, int[][] matrix2,
        int rowLength, int columnLength) {
    int[][] result = new int[rowLength][columnLength];

    for (int x = 0; x < rowLength; x++) {
        for (int y = 0; y < columnLength; y++) {
            result[x][y] = matrix1[x][y] + matrix2[x][y];
        }
    }

    return result;
}

/**
 * Convert the CSV (coma separated values) data to a matrix with float
 * values.
 * @param data the input data with the CSV text.
 * @param attrNum number of attributes (columns) to divide the elements.
 * @return the matrix with the float values.
 */
public static float[][] csvToFloatMatrix(String data, int attrNum){
    final String[] dataRows = data.split(",");
    final int dataRowsLength = dataRows.length;
    float[][] records = new float[dataRowsLength][attrNum];

    for (int y = 0; y < dataRowsLength; y++) {
        final String[] rowData = dataRows[y].split(";");

        for (int x = 0; x < attrNum; x++) {
            records[y][x] = Float.valueOf(rowData[x]);
        }
    }

    return records;
```

```
}

/**
 * Convert the CSV (coma separated values) data to a matrix with integer
 * values.
 * @param data the input data with the CSV text.
 * @param attrNum number of attributes (columns) to divide the elements.
 * @return the matrix with the integer values.
 */
public static int[][] csvToIntMatrix(String data, int attrNum){
   final String[] dataRows = data.split(",");
   final int dataRowsLength = dataRows.length;
   int[][] records = new int[dataRowsLength][attrNum];

   for (int y = 0; y < dataRowsLength; y++) {
      final String[] rowData = dataRows[y].split(";");

      for (int x = 0; x < attrNum; x++) {
         records[y][x] = Integer.valueOf(rowData[x]);
      }
   }

   return records;
}

/**
 * Modify a matrix to the coma separated values format.
 * @param matrix the matrix to be modify.
 * @param rowsNum number of rows in the matrix.
 * @param colsNum number of columns in the matrix.
 * @return a String with the data in the new format.
 */
public static String matrixToCSV(int[][] matrix, int rowsNum, int colsNum){
   StringBuilder csv = new StringBuilder(10000);

   for (int r = 0; r < rowsNum; r++) {
      for (int c = 0; c < colsNum; c++) {
         csv.append(matrix[r][c]);

         if (c != (colsNum - 1)) {
            csv.append(';');
         }
      }
      csv.append("\n");

   }
   return csv.toString();
```

```
    }
}
```

2.4.2. quargFunctions Package

2.4.2.1. quargSoftware.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;

import quargObjects.associationRules;
import quargObjects.dataInput;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;

/**
 * Represents the QUARG algorithm functionality called by the web service.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
@WebService(serviceName = "quargSoftware")
public class quargSoftware {

    /**
     * The function calls the methods to apply the QUARG algorithm with the
     * data that gets as variables.
     * @param dataAttr the list of the attributes names.
     * @param data the list of records.
     * @param antMinComb minimum combination number for the antecedent.
     * @param antMaxComb maximum combination number for the antecedent.
     * @param sucMinComb minimum combination number for the succedent.
     * @param sucMaxComb maximum combination number for the succedent.
     * @param antAttrSel antecedent attributes list.
     * @param sucAttrSel succedent attributes list.
     * @param conf confidence value.
     * @param lift lift value.
     * @param supp support value.
     * @param algorithm name of the algorithm in which the data is going to be
     * analyze ("quarg" = Genetic algorithm, "full" = Full search algorithm).
     * @return the list of the result set of valid rules.
```

```
 */
@WebMethod(operationName = "runQuargAlgorithm")
public String runQuargAlgorithm(
            @WebParam(name = "dataAttr") String dataAttr,
            @WebParam(name = "data") String data,
            @WebParam(name = "antMinComb") int antMinComb,
            @WebParam(name = "antMaxComb") int antMaxComb,
            @WebParam(name = "sucMinComb") int sucMinComb,
            @WebParam(name = "sucMaxComb") int sucMaxComb,
            @WebParam(name = "antAttrSel") String antAttrSel,
            @WebParam(name = "sucAttrSel") String sucAttrSel,
            @WebParam(name = "conf") float conf,
            @WebParam(name = "lift") float lift,
            @WebParam(name = "supp") float supp,
            @WebParam(name = "algorithm") String algorithm){
   final dataInput evaluationData;
   final long start = System.currentTimeMillis();

   try {
      evaluationData = new dataInput(dataAttr, data, antMinComb,
         antMaxComb, sucMinComb, sucMaxComb, antAttrSel, sucAttrSel,
         conf, lift, supp);
   } catch (Exception e) {
      return "There is a problem with the input data, please check it.";
   }
   try {
      final associationRules generatedRules =
            rulesCreation.generateRules(evaluationData);
      if (algorithm.equals("quarg")) { // QUARG algorithm selection.
         final String result =
               areaIdentification.consDifferenceMatrixQUARG(
               evaluationData, generatedRules);
         final long end = System.currentTimeMillis();

         return result + "Execution time was " + (end - start) + " ms.";
      } else if (algorithm.equals("full")) { // Full search selection.
         final String result =
               areaIdentification.consDifferenceMatrixFull(
               evaluationData, generatedRules);
         final long end = System.currentTimeMillis();

         return result + "Execution time was " + (end - start) + " ms.";
      } else {
         return "You have selected an algorithm not implemented yet.\n";
      }
   } catch (Exception e) {
      return "Something wrong happened, check the code!! \n" + e;
```

```
        }
    }

    /**
     * The function calls the methods to apply the preprocessing algorithm to
     * the input data.
     * @param dataAttr the list of the attributes names.
     * @param dataInput the list of records.
     * @param binNum the number of bins or clusters.
     * @param type kind of algorithm to apply (Equi-Depth, Equi-Width, K-Means).
     * @return the data preprocessed.
     */
    @WebMethod(operationName = "discretizeDataInput")
    public String discretizeDataInput(
                @WebParam(name = "dataAttr") String dataAttr,
                @WebParam(name = "dataInput") String dataInput,
                @WebParam(name = "binNum") int binNum,
                @WebParam(name = "type") String type){
        if (type.equals("width")) {
            try {
                return dataPreprocessing.equiWidthAlgorithm(binNum, dataInput,
                        dataAttr);
            } catch (Exception e) {
                return "Something wrong happened, check the code!! \n" + e;
            }
        } else if (type.equals("depth")) {
            try {
                return dataPreprocessing.equiDepthAlgorithm(binNum, dataInput,
                        dataAttr);
            } catch (Exception e) {
                return "Something wrong happened, check the code!! \n" + e;
            }
        } else if (type.equals("kmeans")) {
            try {
                return dataPreprocessing.kMeansAlgorithm(binNum, dataInput,
                        dataAttr);
            } catch (Exception e) {
                return "Something wrong happened, check the code!! \n" + e;
            }
        } else {
            return "You have selected an algorithm not implemented yet.\n";
        }
    }
}
```

2.4.2.2. dataPreprocessing.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;

import commonFunctions.arrays;
import commonFunctions.matrices;
import java.util.Arrays;

/**
 * Handle the functions related with the data discretization method in the
 * application.
 *
 * @version 1.0 Jan 2012
 * @author Adalberto Cubillo
 */
public final class dataPreprocessing {

    /**
     * Discretize the data input using bins with the same range of values.
     * @param numBins number of bins to create.
     * @param dataInput the CSV text for the data input to be discretize.
     * @param dataAttr data input attributes names.
     * @return the String with the CSV text of the data discretized.
     */
    public static String equiWidthAlgorithm(int numBins, String dataInput,
            String dataAttr){
        final int numAttr = dataAttr.split(";").length;
        final float[][] data = matrices.csvToFloatMatrix(dataInput, numAttr);
        final int dataLength = data.length;
        float[] bins = new float[numBins];
        StringBuilder result = new StringBuilder(10000);
        int[][] dataDiscretized = new int[dataLength][numAttr];

        for (int i = 0; i < numAttr; i++) {
            final float[] column = matrices.getColumn(data, i, dataLength);
            final float minVal = arrays.getMinValue(column);
            final float maxVal = arrays.getMaxValue(column);
            final float size = (maxVal - minVal) / numBins;

            for (int b = 1; b < numBins; b++) {
                final float value = minVal + (size * b);
```

```
            bins[b - 1] = value;
        }

        bins[numBins - 1] =  maxVal + 1;

        for (int index = 0; index < dataLength; index++) {
            for (int b = 0; b < numBins; b++) {
                if (column[index] < bins[b]) {
                    dataDiscretized[index][i] = b + 1;
                    break;
                }
            }
        }
    }

    result.append(matrices.matrixToCSV(dataDiscretized, dataLength,
            numAttr));

    return result.toString();
}

/**
 * Discretize the data input using bins with the input values assigned
 * equally to each of them.
 * @param numBins number of bins to create.
 * @param dataInput the CSV text for the data input to be discretize.
 * @param dataAttr data input attributes names.
 * @return the String with the CSV text of the data discretized.
 */
public static String equiDepthAlgorithm(int numBins, String dataInput,
        String dataAttr){
    final int numAttr = dataAttr.split(";").length;
    final float[][] data = matrices.csvToFloatMatrix(dataInput, numAttr);
    final int dataLength = data.length;
    float[] sortedColumn = new float[dataLength];
    float[] bins = new float[numBins];
    int[] binsPositions = new int[numBins];
    StringBuilder result = new StringBuilder(10000);
    int[][] dataDiscretized = new int[dataLength][numAttr];

    for (int b = 1; b <= numBins; b++) {
        final float value = (dataLength * b) / numBins;
        binsPositions[b - 1] = Math.round(value);
    }

    for (int i = 0; i < numAttr; i++) {
        final float[] column = matrices.getColumn(data, i, dataLength);
```

```
        System.arraycopy(column, 0, sortedColumn, 0, dataLength);
        Arrays.sort(sortedColumn);

        for (int b = 0; b < numBins; b++) {
            bins[b] = sortedColumn[binsPositions[b] - 1];
        }

        for (int index = 0; index < dataLength; index++) {
            for (int b = 0; b < numBins; b++) {
                if (column[index] <= bins[b]) {
                    dataDiscretized[index][i] = b + 1;
                    break;
                }
            }
        }
    }

    result.append(matrices.matrixToCSV(dataDiscretized, dataLength,
            numAttr));

    return result.toString();
}

/**
 * Discretize the data input with the K-Means algorithm generating random
 * centroids, an after that adjusting them, to assign the values close to
 * each of them.
 * @param numClusters number of centroids to create.
 * @param dataInput the CSV text for the data input to be discretize.
 * @param dataAttr data input attributes names.
 * @return the String with the CSV text of the data discretized.
 */
public static String kMeansAlgorithm(int numClusters, String dataInput,
        String dataAttr){
    final int numAttr = dataAttr.split(";").length;
    final float[][] data = matrices.csvToFloatMatrix(dataInput, numAttr);
    final int dataLength = data.length;
    StringBuilder result = new StringBuilder(10000);
    int[][] dataDiscretized = new int[dataLength][numAttr];

    for (int i = 0; i < numAttr; i++) {
        final float[] column = matrices.getColumn(data, i, dataLength);
        final float maxVal = arrays.getMaxValue(column);
        float[] centroids = generateCentroids(numClusters, dataLength,
                null, maxVal);
        boolean move = true;
```

```
    float[][] finalClusters = null;

    while (move) {
        float[][] distances = new float[numClusters][dataLength];
        float[][] clusters = new float[numClusters][dataLength];

        /* Obtain the distance between the elements and the centroid. */
        for (int c = 0; c < numClusters; c++) {
            for (int pos = 0; pos < dataLength; pos++) {
                distances[c][pos] = Math.abs(column[pos]
                        - centroids[c]);
            }
        }

        /* Objects clustering. */
        for (int pos = 0; pos < dataLength; pos++) {
            int posMinElement = 0;

            for (int c = 1; c < numClusters; c++) {
                if (distances[posMinElement][pos] > distances[c][pos]) {
                    posMinElement = c;
                }
            }

            clusters[posMinElement][pos] = column[pos];
        }

        if (equalsClusters(finalClusters, clusters, numClusters,
                dataLength)) {
            move = false;
        } else {
            centroids = generateCentroids(numClusters, dataLength,
                    clusters, maxVal);
            finalClusters = clusters;
        }
    }

    for (int index = 0; index < dataLength; index++) {
        for (int c = 0; c < numClusters; c++) {
            if (finalClusters[c][index] != 0) {
                dataDiscretized[index][i] = c + 1;
            }
        }
    }
}

result.append(matrices.matrixToCSV(dataDiscretized, dataLength,
```

```
            numAttr));

    return result.toString();
}

/**
 * Generates the centroids, randomly for the first time, and using the
 * clusters' elements the other times.
 * @param numClusters quantity of centroids.
 * @param numElements quantity of elements in the clusters.
 * @param cluster matrix with the cluster elements to analyze.
 * @param maxVal maximum value that the random generator can take.
 * @return the array with the new centroids.
 */
private static float[] generateCentroids(int numClusters, int numElements,
        float[][] cluster, float maxVal){
    float[] centroids = new float[numClusters];

    if (cluster == null) { // Random generation.
        for (int i = 0; i < numClusters; i++) {
            centroids[i] = (float) Math.random() * maxVal;
        }
    } else { // Determine new centroids.
        for (int c = 0; c < numClusters; c++) {
            float cantElements = 0;
            float sumElements = 0;

            for (int i = 0; i < numElements; i++) {
                float element = cluster[c][i];

                if (element != 0) {
                    cantElements++;
                    sumElements += element;
                }
            }

            centroids[c] = sumElements / cantElements;
        }
    }

    return centroids;
}

/**
 * Determine if the clusters of the previous iteration and the new one are
 * the same.
 * @param clusters1 previous iteration cluster.
```

```
     * @param clusters2 actual iteration cluster.
     * @param numClusters quantity of centroids.
     * @param numElements quantity of elements in the clusters.
     * @return true if the clusters are the same, false otherwise.
     */
    private static boolean equalsClusters(float[][] clusters1,
            float[][] clusters2, int numClusters, int numElements){
        if (clusters1 != null) {
            for (int c = 0; c < numClusters; c++) {
                for (int pos = 0; pos < numElements; pos++) {
                    if (clusters1[c][pos] != clusters2[c][pos]) {
                        return false;
                    }
                }
            }

            return true;
        } else {
            return false;
        }
    }
}
```

2.4.2.3. rulesCreation.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;

import commonFunctions.math;
import java.util.Arrays;
import quargObjects.associationRules;
import quargObjects.dataInput;

/**
 * Contains the functionalities for the creation of the list of rules.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class rulesCreation {

    /**
```

```
 * Generate the rules to be evaluated in the algorithm.
 * @param data dataInput parameter (see dataInput.java).
 * @return a list with the rules.
 */
public static associationRules generateRules(dataInput data) {
   int[] antAttr = null;
   int[] sucAttr = null;
   associationRules result = null;
   final int dataAntMin = data.getAntecedentMin();
   final int dataAntMax = data.getAntecedentMax();
   final int dataSucMin = data.getSuccedentMin();
   final int dataSucMax = data.getSuccedentMax();
   final int dataAttrNum = data.getAttributesNumber();
   final int[] dataAntAttr = data.getAntecedentAttributes();
   final int[] dataSucAttr = data.getSuccedentAttributes();

   /* Number of antecedent elements combinations. */
   for (int antCombIndex = dataAntMin; antCombIndex <= dataAntMax;
         antCombIndex++) {
      final int antAttrCombNum = math.getPower(dataAttrNum, antCombIndex);

      /* Number of succedent elements combinations. */
      for (int sucCombIndex = dataSucMin; sucCombIndex <= dataSucMax;
            sucCombIndex++) {
         final int sucAttrCombNum = math.getPower(dataAttrNum,
               sucCombIndex);

         /* Antecedent attribute number assignations. */
         for (int antAttrComb = 1; antAttrComb <= antAttrCombNum;
               antAttrComb++) {
            antAttr = generateAttrComb(antCombIndex, antAttrComb,
                  dataAttrNum, antAttr);

            if (isUsedAttribute(antAttr, dataAntAttr)) {
               /* Succedent attribute number assignations. */
               for (int sucAttrComb = 1; sucAttrComb <= sucAttrCombNum;
                     sucAttrComb++) {
                  sucAttr = generateAttrComb(sucCombIndex,
                        sucAttrComb, dataAttrNum, sucAttr);

                  if (isUsedAttribute(sucAttr, dataSucAttr)) {
                     if (isValidAttrSet(antCombIndex, sucCombIndex,
                           antAttr, sucAttr)) {
                        final int[] antecedent =
                              generateRecordComb(antCombIndex,
                              antAttr, data);
                        final int[] succedent =
```

```
                        generateRecordComb(sucCombIndex,
                        sucAttr, data);
                    associationRules element =
                        new associationRules(antecedent,
                            succedent, antAttr,
                            sucAttr);

                    if (result == null) {
                       result = element;
                    } else {
                       element.setNext(result);
                       result = element;
                    }
                 }
              }
           }
        }
     }
  }

  return result;
}

/**
 * Generate the antecedent/succedent attributes combinations.
 * @param combIndex Quantity of elements to combine.
 * @param AttrNumComb Attribute number assignation.
 * @param attrNum Attributes quantity.
 * @param attrComb Array which has the previous attribute combination
 * (antecedent/succedent).
 * @return an array with the new attribute combination.
 */
private static int[] generateAttrComb(int combIndex, int AttrNumComb,
      int attrNum, int[] attrComb) {
  if (AttrNumComb == 1) {
     attrComb = new int[combIndex];

     for (int i = 0; i < combIndex; i++) {
        attrComb[i] = 0;
     }
  } else {
     attrComb[0]++;
  }
  for (int i = 0; i < combIndex; i++) {
     if (attrComb[i] >= attrNum) {
        attrComb[i] = 0;
```

```
            attrComb[i + 1]++;
        }
    }

    return attrComb;
}

/**
 * Validate if all the elements of the array are used as a(n)
 * antecedent/succedent.
 * @param arrayEval array with the elements to evaluate.
 * @param arrayComp array with the elements that are antecedent/succedent.
 * @return true if all the elements belongs to the arrayComp,
 * otherwise false.
 */
private static boolean isUsedAttribute(int[] arrayEval, int[] arrayComp) {
    final int evalLength = arrayEval.length;
    final int compLength = arrayComp.length;

    for(int i = 0; i < evalLength; i++) {
        boolean member = false;

        for (int index = 0; index < compLength; index++) {
            if (arrayEval[i] == arrayComp[index]) {
                member = true;
                break;
            }
        }
        if (!member) {
            return false;
        }
    }

    return true;
}

/**
 * Validate that all the attributes in the generated rule accomplish the
 * requirements for combination of attributes in the rules.
 * @param antCombIndex the number of antecedent attributes combinations.
 * @param sucCombIndex the number of succedent attributes combinations.
 * @param antAttr array with the select attributes for the antecedent.
 * @param sucAttr array with the select attributes for the succedent.
 * @return true if the attributes are valid, otherwise false.
 */
private static boolean isValidAttrSet(int antCombIndex, int sucCombIndex,
        int[] antAttr, int[] sucAttr) {
```

```
    /* Antecedent attributes validation. */
    if (validateAttributes(antCombIndex, antAttr)) {

        /* Succedent attributes validation. */
        if (validateAttributes(sucCombIndex, sucAttr)) {
            boolean valid = true;
            final int antAttrLength = antAttr.length;
            final int sucAttrLength = sucAttr.length;
            int[] attrValidation = new int[antAttrLength + sucAttrLength];
            final int attrValidationLength = attrValidation.length - 1;

            System.arraycopy(antAttr, 0, attrValidation, 0, antAttrLength);
            System.arraycopy(sucAttr, 0, attrValidation, antAttrLength,
                    sucAttrLength);
            Arrays.sort(attrValidation);

            /* Validate that none of the attributes are the same. */
            for (int i = 0; i < attrValidationLength; i++) {
                if (attrValidation[i] == attrValidation[i + 1]) {
                    valid = false;
                    break;
                }
            }

            return valid;
        } else {
            return false;
        }
    } else {
        return false;
    }
}

/**
 * Validate that the array with the antecedent/succedent attributes
 * accomplish the requirements for combination of attributes in the rules.
 * @param combIndex the number of attributes combinations.
 * @param attr array with the select attributes
 * @return true if the attributes are valid, otherwise false.
 */
private static boolean validateAttributes(int combIndex, int[] attr){
    boolean valid = true;

    for (int i = 0; i < combIndex; i++) {
        for (int i2 = i; i2 < combIndex; i2++) {
            if (attr[i] < attr[i2]) {
```

```
                valid = false;
                break;
            }
        }
        if (!valid) {
            break;
        }
    }

    return valid;
  }

  /**
   * Generate the antecedent/succedent column combination depending on
   * the attributes combinations.
   * @param CombIndex quantity of elements to combine.
   * @param attrComb indexes in the records to combine.
   * @param data records.
   * @return  an array with the records added (combined).
   */
  private static int[] generateRecordComb(int CombIndex, int[] attrComb,
      dataInput data) {
    final int dataRecNum = data.getRecordsNumber();
    int[] combination = new int[dataRecNum];

    for (int i = 0; i < CombIndex; i++) {
        int[] values = data.getRecordColumn(attrComb[i]);

        /* Add the records. */
        for (int r = 0; r < dataRecNum; r++) {
            combination[r] += values[r];
        }
    }

    return combination;
  }
}
```

2.4.2.4. areaIdentification.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;
```

```
import commonFunctions.arrays;
import commonFunctions.matrices;
import java.util.Arrays;
import quargObjects.associationRules;
import quargObjects.dataInput;
import quargObjects.interestArea;

/**
 * Contains the difference matrix construction and the area identification
 * functionalities.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class areaIdentification {

    /** The antecedent width of the population generation. */
    private static int maxWidthAnt;

    /** The succedent width of the population generation. */
    private static int maxWidthSuc;

    /** List of coordinates for the full search approach. */
    private static int[][] coordinates;

    /** List of values for the full search approach. */
    private static float[] values;

    /** The antecedent higher value in the data input. */
    public static int antMinVal;

    /** The antecedent higher value in the data input. */
    public static int antMaxVal;

    /** The succedent higher value in the data input. */
    public static int sucMinVal;

    /** The succedent higher value in the data input. */
    public static int sucMaxVal;

    /** Stores the quantity of selected rules as result. */
    public static int selectedRulesCount;

    /** Stores the quantity of analyzed rules during the algorithm execution. */
    public static int rulesCount;
```

```
/**
 * Identify the interest areas and evaluate them and determine which rules
 * are valid, this exclusively for the QUARG algorithm approach.
 * @param data the input data.
 * @param rules the generated rules.
 * @return the rules that passed the evaluation.
 */
public static String consDifferenceMatrixQUARG(dataInput data,
        associationRules rules){
    StringBuilder resultBuffer = new StringBuilder(10000);
    selectedRulesCount = 0;
    rulesCount = 0;

    /* Cicle for each of the rules generated before. */
    while (rules != null) {
        antMaxVal = rules.getMaxAntecedentValue();
        sucMaxVal = rules.getMaxSuccedentValue();
        final int[][] actualCount = fillActualCount(data, rules);
        final float[][] expectationCount = fillExpectationCount(data,
                actualCount);
         final float[][] differentialCount = fillDifferentialCount(
                expectationCount, actualCount);
        final int antAttrLength = rules.getAntAttributesLength();
        final int sucAttrLength = rules.getSucAttributesLength();
        final int squareTestVal = (antAttrLength + sucAttrLength) * 2;
        maxWidthAnt = Math.max(Math.round((antAttrLength * antMaxVal) / 5),
                2);
        maxWidthSuc = Math.max(Math.round(sucAttrLength * sucMaxVal / 5),
                2);

        /* Genetic algorithm method call. */
        final interestArea intArea = identifyInterestAreaQUARG(
                squareTestVal, differentialCount, antAttrLength,
                sucAttrLength);

        resultBuffer.append('|');
        resultBuffer.append(generalRuleResult(data, rules));

        /* Area decomposition method call */
        final String rule = areaDecomposition.decomposeArea(intArea, data,
                rules);

        resultBuffer.append(rule);
        rules = rules.getNext();
    }

    resultBuffer.append("|Number of rules verifications: ");
```

```
    resultBuffer.append(rulesCount);
    resultBuffer.append(", selected rules: ");
    resultBuffer.append(selectedRulesCount);
    resultBuffer.append(".\n");

    return resultBuffer.toString();
}

/**
 * Fill the actual count matrix with the data that corresponds.
 * @param data the input data.
 * @param rules the generated rules.
 * @return the actual count matrix.
 */
private static int[][] fillActualCount(dataInput data,
        associationRules rules) {
    int[][] matrixCount = new int[antMaxVal][sucMaxVal];
    final int dataRecNum = data.getRecordsNumber();
    for (int i = 0; i < dataRecNum; i++) {
        /* the -1 is because the index begins in 0, and the value is
         * from 1 to N. */
        final int posX = rules.getAntecedentElement(i) - 1;
        final int posY = rules.getSuccedentElement(i) - 1;
        matrixCount[posX][posY]++;
    }

    return matrixCount;
}

/**
 * Fill the expectative count matrix with the data that corresponds.
 * @param data the input data.
 * @param actualCount the actual count matrix.
 * @return the expectation count matrix.
 */
private static float[][] fillExpectationCount(dataInput data,
        int[][] actualCount) {
    float[][] expectationMatrix = new float[antMaxVal][sucMaxVal];
    final float recNum = data.getRecordsNumber();

    for (int x = 0; x < antMaxVal; x++) {
        for (int y = 0; y < sucMaxVal; y++) {
            final float columnSum = matrices.sumColumnElements(
                        actualCount, y, antMaxVal);
            final float rowSum = matrices.sumRowElements(actualCount,
                x, sucMaxVal);
            expectationMatrix[x][y] = (columnSum * rowSum) / recNum;
```

```
        }
    }
    return expectationMatrix;
}

/**
 * Fill the differential count matrix with the data that corresponds.
 * @param expectationCount the expectation count matrix.
 * @param actualCount the actual count matrix.
 * @return the differential count matrix.
 */
private static float[][] fillDifferentialCount(float[][] expectationCount,
        int[][] actualCount) {
    float[][] differentialMatrix = new float[antMaxVal][sucMaxVal];

    for (int x = 0; x < antMaxVal; x++) {
        for (int y = 0; y < sucMaxVal; y++) {
            differentialMatrix[x][y] = ((float) actualCount[x][y])
                    - expectationCount[x][y];
        }
    }
    return differentialMatrix;
}

/**
 * Identify the areas with the most probability of occurrence in a rule,
 * using a genetic mutation algorithm.
 * @param squareTestVal quantity of squares coordinates.
 * @param differentialCount matrix with the percentage of occurrence of
 * the value in the rule.
 * @param antComb the quantity of attributes that forms the antecedent.
 * @param sucComb the quantity of attributes that forms the succedent.
 * @return an object with the matrix coordinates and quantity of squares.
 */
private static interestArea identifyInterestAreaQUARG(int squareTestVal,
        float[][] differentialCount, int antComb,int sucComb) {
    /* Settings variables. */
    final int genNum = (antComb + sucComb) * 3;
    final int chromeLength = 4;
    final int populationSize = (antMaxVal + sucMaxVal) * 2;

    /* Genetic algorithm tables. */
    int[][] population = new int[populationSize][chromeLength];
    int[][] newPopulation = new int[populationSize][chromeLength];
    int[][] finalArea = new int[antMaxVal][sucMaxVal];
    int[][] finalCoordinates = new int[squareTestVal][chromeLength];
```

```
/* Mutation variables. */
float[] fitness = new float[populationSize];
int[] indexFitness = new int[populationSize];
int squaresNum = 0;

/* Random population generation. */
population = setPopulation(population, 0, populationSize);

for (int gen = 0; gen < genNum; gen++) {
   int indivNum = 0;

   /* Population fitness. */
   fitness = setFitness(fitness, population, populationSize,
         differentialCount, antComb);

   final float fitnessMaxVal = arrays.getMaxValue(fitness);

   /* Selection of certain elements in population. */
   for (int chrom = 0; chrom < populationSize; chrom++) {
      if (fitness[chrom] > (fitnessMaxVal / 2)) {
         /* Copy the whole row. */
          newPopulation[indivNum] = population[chrom];
          indivNum++;
      }
   }

   /* Mutation of the population. */
   newPopulation = mutatePopulation(newPopulation, indivNum,
         chromeLength);
   /* Filling the population after the mutation. */
   population = setPopulation(newPopulation, indivNum, populationSize);
}

indexFitness = arrays.sortIndex(fitness, "descend");

/* Evaluate the squares coordinates. */
for (int squares = 0; squares < squareTestVal; squares++) {
   int overlapping = 0;
   int[][] suppFinalArea = new int[antMaxVal][sucMaxVal];

   for (int x = population[indexFitness[squares]][0];
         x <= population[indexFitness[squares]][1]; x++) {
      for (int y = population[indexFitness[squares]][2];
            y <= population[indexFitness[squares]][3]; y++) {
         suppFinalArea[x - 1][y - 1]++;
         /* -1 is because the index begins in 0,
          * and the value is from 1 to N. */
```

```
            }
        }

        for (int x = 0; x < antMaxVal; x++) {
            for (int y = 0; y < sucMaxVal; y++) {
                if ((finalArea[x][y] >= 1) && (suppFinalArea[x][y] >= 1)) {
                    overlapping++;
                }
            }
        }

        final int popAntDif = population[indexFitness[squares]][1]
                - population[indexFitness[squares]][0];
        final int popSucDif = population[indexFitness[squares]][2]
                - population[indexFitness[squares]][3];
        final int squareSizeUpdate = Math.abs(popAntDif)
                + Math.abs(popSucDif);

        if ((squares == 0) || ((squares > 0)
                && (squareSizeUpdate > (0.5 * overlapping)))) {
            finalCoordinates[squaresNum][0] =
                    population[indexFitness[squares]][0];
            finalCoordinates[squaresNum][1] =
                    population[indexFitness[squares]][1];
            finalCoordinates[squaresNum][2] =
                    population[indexFitness[squares]][2];
            finalCoordinates[squaresNum][3] =
                    population[indexFitness[squares]][3];
            finalArea = matrices.sumMatrices(finalArea,
                    suppFinalArea, antMaxVal, sucMaxVal);
            squaresNum++;
        }
    }

    return new interestArea(squaresNum, finalCoordinates);
}

/**
 * Set random values to the elements of the population matrix, for the
 * first new generation.
 * @param pMatrix population matrix.
 * @param pInitialPosition the row in which start setting the
 * population values.
 * @param pSize population size.
 * @return the matrix with the new elements.
 */
private static int[][] setPopulation(int[][] pMatrix, int pInitialPosition,
```

```
        int pSize) {
    for (int i = pInitialPosition; i < pSize; i++) {
        pMatrix[i][0] = (int) Math.ceil(Math.random() * antMaxVal);
        pMatrix[i][1] = (int) Math.ceil(Math.random() * (maxWidthAnt + 1))
                + pMatrix[i][0];

        if (pMatrix[i][1] > antMaxVal) {
            pMatrix[i][1] = antMaxVal;
        }

        pMatrix[i][2] = (int) Math.ceil(Math.random() * sucMaxVal);
        pMatrix[i][3] = (int) Math.ceil(Math.random() * (maxWidthSuc + 1))
                + pMatrix[i][2];

        if (pMatrix[i][3] > sucMaxVal) {
            pMatrix[i][3] = sucMaxVal;
        }
    }

    return pMatrix;
}

/**
 * Generates an array with the fitness values for the population.
 * @param fitness the fitness array.
 * @param population the population array.
 * @param populationSize the size of the population.
 * @param differentialCount the differential count array.
 * @param antComb the actual number of antecedent combination.
 * @return an array with the fitness values.
 */
private static float[] setFitness(float[] fitness, int[][] population,
        int populationSize, float[][] differentialCount, int antComb) {
    for (int chrom = 0; chrom < populationSize; chrom++) {
        final int antMinCoord = population[chrom][0];
        final int antMaxCoord = population[chrom][1];
        final int sucMinCoord = population[chrom][2];
        final int sucMaxCoord = population[chrom][3];
        float sum = 0;

        for (int y = antMinCoord; y <= antMaxCoord; y++) {
            for (int x = sucMinCoord; x <= sucMaxCoord; x++) {
                sum += differentialCount[y - 1][x - 1];
                /* the -1 is because the index begins in 0,
                 * and the value is from 1 to N. */
            }
        }
```

```
        if ((antMaxCoord - antMinCoord) < antComb) {
            fitness[chrom] = -10000;
        } else {
            fitness[chrom] = sum;
        }
    }

    return fitness;
}

/**
 * Mutate the population elements with random values.
 * @param newPopulation the population matrix.
 * @param indivNum maximum population rows.
 * @param chromeLength chromosome maximum value.
 * @return the population matrix.
 */
private static int[][] mutatePopulation(int[][] newPopulation,
        int indivNum, int chromeLength) {
    int mutation = (int) Math.ceil(Math.random() * indivNum);
    int mutationNum = (int) Math.ceil(Math.random() * chromeLength);

    if ((mutation >= indivNum) && (indivNum != 0)) {
        mutation = indivNum - 1; // -1 is for fitting the index.
    }
    if (mutationNum == chromeLength) {
        mutationNum = chromeLength - 1; // -1 is for fitting the index.
    }

    if (mutationNum == 0) {
        newPopulation[mutation][mutationNum] = (int) Math.ceil(Math.random()
                * antMaxVal);

        if (newPopulation[mutation][mutationNum]
                > newPopulation[mutation][1]) {
            newPopulation[mutation][mutationNum] =
                    newPopulation[mutation][1];
        }
    } else if (mutationNum == 1) {
        newPopulation[mutation][mutationNum] = (int) Math.ceil(Math.random()
                * antMaxVal);

        if (newPopulation[mutation][mutationNum]
                < newPopulation[mutation][0]) {
            newPopulation[mutation][mutationNum] =
                    newPopulation[mutation][0];
```

```
        }
    } else if(mutationNum == 2) {
        newPopulation[mutation][mutationNum] = (int) Math.ceil(Math.random()
                * sucMaxVal);

        if (newPopulation[mutation][mutationNum]
                > newPopulation[mutation][3]) {
            newPopulation[mutation][mutationNum] =
                    newPopulation[mutation][3];
        }
    } else if (mutationNum == 3) {
        newPopulation[mutation][mutationNum] = (int) Math.ceil(Math.random()
                * sucMaxVal);

        if (newPopulation[mutation][mutationNum]
                < newPopulation[mutation][2]) {
            newPopulation[mutation][mutationNum] =
                    newPopulation[mutation][2];
        }
    }

    return newPopulation;
}

/**
 * Identify the interest areas and evaluate them and determine which rules
 * are valid, this exclusively for the full search approach.
 * @param data the input data.
 * @param rules the generated rules.
 * @return the rules that passed the evaluation.
 */
public static String consDifferenceMatrixFull(dataInput data,
        associationRules rules){
    StringBuilder resultBuffer = new StringBuilder(10000);
    selectedRulesCount = 0;
    rulesCount = 0;

    /* Cicle for each of the rules generated before. */
    while (rules != null) {
        final int[] antRecIndex =
                generateRecordsIndexation(rules.getAntecedent());
        final int[] sucRecIndex =
                generateRecordsIndexation(rules.getSuccedent());
        antMaxVal = antRecIndex.length;
        sucMaxVal = sucRecIndex.length;
        final int[][] actualCount = fillActualCountAlt(data, rules,
                antRecIndex, sucRecIndex);
```

```
        final float[][] expectationCount = fillExpectationCount(data,
                actualCount);
        final float[][] differentialCount = fillDifferentialCount(
                expectationCount, actualCount);

        final interestArea intArea = identifyInterestAreaFull(
                differentialCount, antRecIndex, sucRecIndex);

        resultBuffer.append('|');
        resultBuffer.append(generalRuleResult(data, rules));

        /* Area decomposition method call */
        final String rule = areaDecomposition.decomposeArea(intArea, data,
                rules);

        resultBuffer.append(rule);
        rules = rules.getNext();
    }

    resultBuffer.append("|\nNumber of rules verifications: ");
    resultBuffer.append(rulesCount);
    resultBuffer.append(", selected rules: ");
    resultBuffer.append(selectedRulesCount);
    resultBuffer.append(".\n");

    return resultBuffer.toString();
}

/**
 * Generate an array of indexes for the records elements, allowing simplify
 * and boost the records analysis.
 * @param records matrix with the data input records.
 * @return an array with the non repeated records elements for the
 * antecedent/succedent.
 */
private static int[] generateRecordsIndexation(int[] records) {
    final int recordsLength = records.length;
    int[] indexation = new int[1];
    int[] recordsCopy = Arrays.copyOf(records, recordsLength);
    Arrays.sort(recordsCopy);
    indexation[0] = recordsCopy[0];

    for (int i = 1; i < recordsLength; i++) {
        final int indexLength = indexation.length;
        if (recordsCopy[i] != indexation[indexLength - 1]) {
            int[] tempIndexation = new int[indexLength + 1];
            System.arraycopy(indexation, 0, tempIndexation, 0,
```

```
                indexLength);
            tempIndexation[indexLength] = recordsCopy[i];
            indexation = tempIndexation;
        }
    }

    return indexation;
}

/**
 * Fill the matrix with the count of elements for the rule being analyzed.
 * @param data data input values.
 * @param rules the rule being analyzed.
 * @param antRecordIndex the antecedent records indexation.
 * @param sucRecordIndex the succedent records indexation.
 * @return the actual count matrix.
 */
private static int[][] fillActualCountAlt(dataInput data,
        associationRules rules, int[] antRecordIndex,
        int[] sucRecordIndex) {
    final int dataRecNum = data.getRecordsNumber();
    int[][] matrixCount = new int[antMaxVal][sucMaxVal];

    for (int i = 0; i < dataRecNum; i++) {
        final int posX = Arrays.binarySearch(antRecordIndex,
                rules.getAntecedentElement(i));
        final int posY = Arrays.binarySearch(sucRecordIndex,
                rules.getSuccedentElement(i));
        matrixCount[posX][posY]++;
    }

    return matrixCount;
}

/**
 * Identify the areas with the most probability of occurrence in a rule,
 * using the full search algorithm.
 * @param differentialCount matrix with the percentage of occurrence of
 * the value in the rule.
 * @param antRecIndex the array with the antecedent indexation.
 * @param sucRecIndex the array with the succedent indexation.
 * @return an object with the matrix coordinates and quantity of squares.
 */
private static interestArea identifyInterestAreaFull(
        float[][] differentialCount, int[] antRecIndex, int[] sucRecIndex) {
    coordinates = null;
    values = null;
```

```
   for (int posIniX = 0; posIniX < antMaxVal; posIniX++) {
      for (int posIniY = 0; posIniY < sucMaxVal; posIniY++) {
         for (int posFinX = posIniX; posFinX < antMaxVal; posFinX++) {
            for (int posFinY = posIniY; posFinY < sucMaxVal;
                  posFinY++) {
               final float value =
                     getCoordinatesValue(differentialCount,  posIniX,
                     posFinX, posIniY, posFinY);
               final int[] coordinate = {posIniX, posFinX, posIniY,
                  posFinY};

               if (value > 0) {
                  if (coordinates == null) {
                     coordinates = new int[1][4];
                     coordinates[0] = coordinate;
                     values = new float[1];
                     values[0] = value;
                  } else {
                     final boolean overlapping = findOverlap(value,
                           coordinate);

                     if (!overlapping) {
                        addCoordinates(coordinate, value);
                     }
                  }
               }
            }
         }
      }
   }

   if (coordinates != null) {
      final int[][] finalCoordinates = getRealCoordinates(antRecIndex,
            sucRecIndex);

      return new interestArea(finalCoordinates.length, finalCoordinates);
   } else {
      return new interestArea(0, null);
   }
}

/**
 * Get the value of the sum of elements in the differential count matrix,
 * this for an specific area in the matrix.
 * @param differentialCount the differential count matrix.
 * @param posIniX initial position for the antecedent in the matrix.
```

```
 * @param posFinX final position for the antecedent in the matrix.
 * @param posIniY initial position for the succedent in the matrix.
 * @param posFinY final position for the succedent in the matrix.
 * @return the sum value of the elements.
 */
private static float getCoordinatesValue(float[][] differentialCount,
      int posIniX, int posFinX, int posIniY, int posFinY) {
   float value = 0;

   for (int index = posIniX; index <= posFinX; index++) {
      final float[] row = Arrays.copyOfRange(differentialCount[index],
            posIniY, posFinY + 1);
      value += arrays.sumArrayElements(row);
   }

   return value;
}

/**
 * Find between elements already selected an a new one coordinate if there
 * is overlap, if it does analyze the validity of the new coordinate and
 * add it to the list.
 * @param value the value of the elements in the new coordinate.
 * @param coordinate the array with the positions of the coordinate in the
 * matrix.
 * @return true if find overlap, otherwise false.
 */
private static boolean findOverlap(float value, int[] coordinate) {
   final int coordLength = coordinates.length;
   boolean overlap = false;
   boolean valid = true;

   for (int i = 0; i < coordLength; i++) {
      final int[] actualCoordinate = coordinates[i];
      final float actualValue = values[i];

      if (((coordinate[0] >= actualCoordinate[0]
            && coordinate[0] <= actualCoordinate[1])
            || (coordinate[1] >= actualCoordinate[0]
            && coordinate[1] <= actualCoordinate[1]))
            && ((coordinate[2] >= actualCoordinate[2]
            && coordinate[2] <= actualCoordinate[3])
            || (coordinate[3] >= actualCoordinate[2]
            && coordinate[3] <= actualCoordinate[3]))) {
         overlap = true;
         final float overlapArea = getOverlapArea(coordinate,
               actualCoordinate);
```

```
            final float area = (Math.abs(coordinate[0] - coordinate[1]) + 1)
                  * (Math.abs(coordinate[2] - coordinate[3]) + 1);
            final float areaDiff = overlapArea / area;
            final float valueDiff = value / actualValue;

            if (areaDiff > valueDiff) {
               valid = false;
               break;
            }
         }
      }

      if (overlap && valid) {
         addCoordinates(coordinate, value);
      }

      return overlap;
   }

   /**
    * Determine how many elements conforms the overlap area.
    * @param coordinate the new coordinate being analyzed.
    * @param actualCoordinate the coordinate for the comparison.
    * @return the number of elements (positions) in the matrix that conforms
    * the overlap.
    */
   private static float getOverlapArea(int[] coordinate,
         int[] actualCoordinate) {
      final int minValX = Math.min(coordinate[0], actualCoordinate[0]);
      final int maxValX = Math.max(coordinate[1], actualCoordinate[1]);
      final int minValY = Math.min(coordinate[2], actualCoordinate[2]);
      final int maxValY = Math.max(coordinate[3], actualCoordinate[3]);
      int valX = 0;
      int valY = 0;

      for (int x = minValX; x <= maxValX; x++) {
         if ((x >= actualCoordinate[0] && x <= actualCoordinate[1])
               && (x >= coordinate[0] && x <= coordinate[1])) {
            valX++;
         }
      }

      for (int y = minValY; y <= maxValY; y++) {
         if ((y >= actualCoordinate[2] && y <= actualCoordinate[3])
               && (y >= coordinate[2] && y <= coordinate[3])) {
            valY++;
         }
```

```
    }

    return (valX * valY);
}

/**
 * Add a coordinate to the list of accepted coordinates, and it's
 * corresponded value to the values list.
 * @param coordinate the new coordinate.
 * @param value the value to add to the list of values.
 */
private static void addCoordinates(int[] coordinate, float value) {
    final int coordLength = coordinates.length;
    final int valLength = values.length;
    int[][] tempCoordinates = new int[coordLength + 1][4];
    System.arraycopy(coordinates, 0, tempCoordinates, 0,
            coordLength);
    tempCoordinates[coordLength] = coordinate;
    coordinates = tempCoordinates;

    float[] tempValues = new float[valLength + 1];
    System.arraycopy(values, 0, tempValues, 0, valLength);
    tempValues[valLength] = value;
    values = tempValues;
}

/**
 * Change the indexed positions for the real records positions in the matrix
 * of the valid coordinates.
 * @param antRecIndex the array with the antecedent indexation.
 * @param sucRecIndex the array with the succedent indexation.
 * @return the matrix with the real positions.
 */
private static int[][] getRealCoordinates(int[] antRecIndex,
        int[] sucRecIndex) {
    final int coordLength = coordinates.length;
    int[][] finalCoordinates = new int[coordLength][4];

    for (int x = 0; x < coordLength; x++) {
        final int[] coordinate = {antRecIndex[coordinates[x][0]],
            antRecIndex[coordinates[x][1]], sucRecIndex[coordinates[x][2]],
            sucRecIndex[coordinates[x][3]]};

        finalCoordinates[x] = coordinate;
    }

    return finalCoordinates;
```

```
    }

    /**
     * Generate the general rule compound result.
     * @param data the data input.
     * @param rules the rule being analyze.
     * @return the general rule structure.
     */
    private static String generalRuleResult(dataInput data,
            associationRules rules){
        final int[] rulesAntAttr = rules.getAntAttributes();
        final int[] rulesSucAttr = rules.getSucAttributes();
        final int antAttrLength = rules.getAntAttributesLength();
        final int sucAttrLength = rules.getSucAttributesLength();
        StringBuilder ruleData = new StringBuilder(50);

        for (int index = 0; index < antAttrLength; index++) {
            final String attrName =
                    data.getAttributesElement(rulesAntAttr[index]);

            ruleData.append(attrName);
            ruleData.append(" ");
        }

        ruleData.append("--> ");

        for (int index = 0; index < sucAttrLength; index++) {
            final String attrName =
                    data.getAttributesElement(rulesSucAttr[index]);
            ruleData.append(attrName);
            ruleData.append(" ");
        }

        return ruleData.toString();
    }
}
```

2.4.2.5. areaDecomposition.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;
```

```
import commonFunctions.arrays;
import java.util.Arrays;
import quargObjects.analyzedRule;
import quargObjects.associationRules;
import quargObjects.dataInput;
import quargObjects.decomposedArea;
import quargObjects.interestArea;

/**
 * Contains the functionalities related to the rules area decomposition.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class areaDecomposition {

    /** Object that stores the decomposed area values. */
    private static decomposedArea decArea;

    /** Quantity of antecedent attributes elements. */
    private static int antAttrNum;

    /** Quantity of succedent attributes elements. */
    private static int sucAttrNum;

    /**
     * Decompose the area of interest for each rule and generates some
     * validations, depending on the amount of interest areas per rule.
     * @param intArea interest area values of a certain rule.
     * @param data the data input values of the application.
     * @param rules the generated rule which is being evaluated.
     * @return the valid rules.
     */
    public static String decomposeArea(interestArea intArea, dataInput data,
            associationRules rules) {
        final int squareNum = intArea.getSquaresNum();
        final int rulesAntAttrLength = rules.getAntAttributesLength();
        final int rulesSucAttrLength = rules.getSucAttributesLength();
        StringBuilder resultBuffer = new StringBuilder(4000);
        antAttrNum = rules.getAntAttributesLength();
        sucAttrNum = rules.getSucAttributesLength();

        for (int cont = 0; cont < squareNum; cont++) {
            decArea = new decomposedArea();

            /** Antecedent decomposition. */
            if (rulesAntAttrLength == 1) {
```

```
    decArea.incrementAntNumber();
    int[][] tempMatrix = new int[1][2];
    tempMatrix[0][0] = Math.min(
            intArea.getCoordinatesElement(cont, 0),
            intArea.getCoordinatesElement(cont, 1));
    tempMatrix[0][1] = Math.max(
            intArea.getCoordinatesElement(cont, 0),
            intArea.getCoordinatesElement(cont, 1));
    decArea.setDecompAntecedent(tempMatrix);
} else {
    startAntDecomposition(new int[antAttrNum],
            1, intArea.getCoordinatesElement(cont, 0),
            intArea.getCoordinatesElement(cont, 1));
    if (decArea.getDecompAntNumber() == 0) {
        int[][] tempMatrix = new int[1][antAttrNum * 2];
        Arrays.fill(tempMatrix[0], 1);
        decArea.setDecompAntecedent(tempMatrix);
        decArea.incrementAntNumber();
    }
}

/** Succedent decomposition. */
if (rulesSucAttrLength == 1) {
    decArea.incrementSucNumber();
    int[][] tempMatrix = new int[1][2];
    tempMatrix[0][0] = Math.min(
            intArea.getCoordinatesElement(cont, 2),
            intArea.getCoordinatesElement(cont, 3));
    tempMatrix[0][1] = Math.max(
            intArea.getCoordinatesElement(cont, 2),
            intArea.getCoordinatesElement(cont, 3));
    decArea.setDecompSuccedent(tempMatrix);
} else {
    startSucDecomposition(new int[sucAttrNum],
            1, intArea.getCoordinatesElement(cont, 2),
            intArea.getCoordinatesElement(cont, 3));
    if (decArea.getDecompSucNumber() == 0) {
        int[][] tempMatrix = new int[1][sucAttrNum * 2];
        Arrays.fill(tempMatrix[0], 1);
        decArea.setDecompSuccedent(tempMatrix);
        decArea.incrementSucNumber();
    }
}

/* Rule selection */
final analyzedRule rule = ruleSelection.selectRule(decArea, data,
        rules);
```

```
        resultBuffer.append(rule.getRuleResult());
        areaIdentification.rulesCount += rule.getVerificationsNumber();
        areaIdentification.selectedRulesCount += rule.getRulesNumber();
    }

    return resultBuffer.toString();
}

/**
 * Recursive method that decompose the antecedent part of the rule, start
 * generating the first part of the decomposition.
 * @param combArray array with the first part of the decomposition.
 * @param cyclesCount counts the number of recursive calls.
 * @param initialAnt validation value to compare the first (start) part of
 * the decomposition.
 * @param finalAnt validation value to compare the second part (stop) of
 * the decomposition.
 */
private static void startAntDecomposition(int[] combArray,int cyclesCount,
        int initialAnt, int finalAnt) {
    for (int start = 1; start <= areaIdentification.antMaxVal; start++) {
        if (cyclesCount == antAttrNum) {
            combArray[cyclesCount - 1] = start;
            if (arrays.sumArrayElements(combArray) == initialAnt) {
                stopAntDecomposition(combArray, new int[antAttrNum], 1,
                        finalAnt);
            }
        } else {
            combArray[cyclesCount - 1] = start;
            startAntDecomposition(combArray, cyclesCount + 1, initialAnt,
                    finalAnt);
        }
    }
}

/**
 * Recursive method that decompose the antecedent part of the rule, finish
 * generating the second part of the decomposition.
 * @param combArrayInit array with the first part of the decomposition.
 * @param combArrayFin array with the second part of the decomposition.
 * @param cyclesCount counts the number of recursive calls.
 * @param finalAnt validation value to compare the second part (stop) of
 * the decomposition.
 */
private static void stopAntDecomposition(int[] combArrayInit,
        int[] combArrayFin, int cyclesCount, int finalAnt) {
```

```
    final int limit = Math.min((combArrayInit[cyclesCount - 1] + 1),
            areaIdentification.antMaxVal);

    for (int stop = combArrayInit[cyclesCount - 1]; stop <= limit; stop++) {
        if (cyclesCount == antAttrNum) {
            combArrayFin[cyclesCount - 1] = stop;
            if (arrays.sumArrayElements(combArrayFin)
                    == finalAnt) {
                int[] tempArray = new int[antAttrNum * 2];

                System.arraycopy(combArrayInit, 0, tempArray, 0,
                        antAttrNum);
                System.arraycopy(combArrayFin, 0, tempArray, antAttrNum,
                        antAttrNum);
                int[][] tempMatrix = modifyDecompMatrix(
                        decArea.getDecompAntecedent(),
                        decArea.getDecompAntNumber(), tempArray);
                decArea.setDecompAntecedent(tempMatrix);
                decArea.incrementAntNumber();
            }
        } else {
            combArrayFin[cyclesCount - 1] = stop;
            stopAntDecomposition(combArrayInit, combArrayFin,
                    cyclesCount + 1, finalAnt);
        }
    }
}

/**
 * Recursive method that decompose the succedent part of the rule, start
 * generating the first part of the decomposition.
 * @param combArray array with the first part of the decomposition.
 * @param cyclesCount counts the number of recursive calls.
 * @param initialSuc validation value to compare the first (start) part of
 * the decomposition.
 * @param finalSuc validation value to compare the second part (stop) of
 * the decomposition.
 */
private static void startSucDecomposition(int[] combArray,int cyclesCount,
        int initialSuc, int finalSuc) {
    for (int start = 1; start <= areaIdentification.sucMaxVal; start++) {
        if (cyclesCount == sucAttrNum) {
            combArray[cyclesCount - 1] = start;
            if (arrays.sumArrayElements(combArray) == initialSuc) {
                stopSucDecomposition(combArray, new int[sucAttrNum], 1,
                        finalSuc);
            }
```

```
        } else {
            combArray[cyclesCount - 1] = start;
            startSucDecomposition(combArray, cyclesCount + 1, initialSuc,
                    finalSuc);
        }
    }
}

/**
 * Recursive method that decompose the succedent part of the rule, finish
 * generating the second part of the decomposition.
 * @param combArrayInit array with the first part of the decomposition.
 * @param combArrayFin array with the second part of the decomposition.
 * @param cyclesCount counts the number of recursive calls.
 * @param finalSuc validation value to compare the second part (stop) of
 * the decomposition.
 */
private static void stopSucDecomposition(int[] combArrayInit,
        int[] combArrayFin, int cyclesCount, int finalSuc) {
    final int limit = Math.min((combArrayInit[cyclesCount - 1] + 1),
            areaIdentification.sucMaxVal);

    for (int stop = combArrayInit[cyclesCount - 1]; stop <= limit; stop++) {
        if (cyclesCount == sucAttrNum) {
            combArrayFin[cyclesCount - 1] = stop;
            if (arrays.sumArrayElements(combArrayFin)
                    == finalSuc) {
                int[] tempArray = new int[sucAttrNum * 2];

                System.arraycopy(combArrayInit, 0, tempArray, 0,
                        sucAttrNum);
                System.arraycopy(combArrayFin, 0, tempArray, sucAttrNum,
                        sucAttrNum);
                int[][] tempMatrix = modifyDecompMatrix(
                        decArea.getDecompSuccedent(),
                        decArea.getDecompSucNumber(), tempArray);
                decArea.setDecompSuccedent(tempMatrix);
                decArea.incrementSucNumber();
            }
        } else {
            combArrayFin[cyclesCount - 1] = stop;
            stopSucDecomposition(combArrayInit, combArrayFin,
                    cyclesCount + 1, finalSuc);
        }
    }
}
```

```
    /**
     * Dynamically modifies the size of the antecedent/succedent
     * decomposition matrix.
     * @param decMatrix the decomposition matrix to be modified.
     * @param decIndex the number of rows in the decomposition matrix.
     * @param tempArray the array to be added to the matrix.
     * @return the decomposition matrix modified.
     */
    private static int[][] modifyDecompMatrix(int[][] decMatrix, int decIndex,
            int[] tempArray) {
        int[][] tempMatrix = new int[decIndex + 1][tempArray.length];
        if (decMatrix == null) {
            tempMatrix[decIndex] = tempArray;
        } else {
            System.arraycopy(decMatrix, 0, tempMatrix, 0,
                    decIndex);
            tempMatrix[decIndex] = tempArray;
        }

        return tempMatrix;
    }
}
```

2.4.2.6. ruleSelection.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargFunctions;

import quargObjects.analyzedRule;
import quargObjects.associationRules;
import quargObjects.dataInput;
import quargObjects.decomposedArea;

/**
 * Contains the rule validation functionalities.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class ruleSelection {

    /**
```

```
 * Evaluate the area decomposed of the rule to make it a valid result.
 * @param decArea decomposed area.
 * @param data input data of the algorithm.
 * @param rules the rule that is being evaluated.
 * @return an object that contain the valid rules and the count of the
 * evaluated elements and the valid elements.
 */
public static analyzedRule selectRule(decomposedArea decArea,
        dataInput data, associationRules rules) {
    analyzedRule result = new analyzedRule();
    boolean validRule = false;
    int initalVal = 0;
    int finalVal = 0;
    String resultConf = "";
    String resultSupp = "";
    String resultLift = "";
    final int antAttrLength = rules.getAntAttributesLength();
    final int sucAttrLength = rules.getSucAttributesLength();
    final int decAreaDecAntNum = decArea.getDecompAntNumber();
    final int decAreaDecSucNum = decArea.getDecompSucNumber();
    final int dataRecNum = data.getRecordsNumber();
    final int[][] dataRec = data.getRecords();
    final int[] rulesAntAttr = rules.getAntAttributes();
    final int[] rulesSucAttr = rules.getSucAttributes();
    final int[][] decAreaDecAnt = decArea.getDecompAntecedent();
    final int[][] decAreaDecSuc = decArea.getDecompSuccedent();
    final float dataConf = data.getConf();
    final float dataSupp = data.getSupp();
    final float dataLift = data.getLift();

    for (int i = 0; i < decAreaDecAntNum; i++) {
        for (int j = 0; j < decAreaDecSucNum; j++) {
            result.incrementVerificationsNumber();
            int AntCompSucComp = 0;
            int AntUncompSucComp = 0;
            int AntCompSucUncomp = 0;

            for (int record = 0; record < dataRecNum;
                    record++) {

                /** Antecedent testing. */
                final boolean antTestValid = testAttributes(antAttrLength,
                        dataRec, record, rulesAntAttr, decAreaDecAnt, i);

                /** Succedent testing. */
                final boolean sucTestValid = testAttributes(sucAttrLength,
                        dataRec, record, rulesSucAttr, decAreaDecSuc, j);
```

```
            /** Rule attributes combination verification. */
            if ((antTestValid) && (sucTestValid)) {
               AntCompSucComp++;
            } else if ((antTestValid) && (!sucTestValid)) {
               AntCompSucUncomp++;
            } else if ((!antTestValid) && (sucTestValid)) {
               AntUncompSucComp++;
            }
         }

         final float support = getRuleSupport(AntCompSucComp,
               dataRecNum);
         final float confidence = getRuleConfidence(AntCompSucComp,
               AntCompSucUncomp);
         final float lift = getRuleLift(AntCompSucComp, AntCompSucUncomp,
                AntUncompSucComp, dataRecNum);

         if ((confidence > dataConf) && (support > dataSupp)
               && (lift > dataLift)) {
            validRule = true;
            initalVal = i;
            finalVal = j;
            resultConf = shortenValue(confidence);
            resultSupp = shortenValue(support);
            resultLift = shortenValue(lift);
         }
      }
   }

   if (validRule) {
      result.incrementRulesNumber();
      StringBuilder ruleData = new StringBuilder(2000);
      ruleData.append('*');

      /** Antecedent result creation. */
      ruleData.append(createResult(antAttrLength, data, rulesAntAttr,
            decAreaDecAnt, initalVal));

      ruleData.append("--> ");

      /** Succedent result creation. */
      ruleData.append(createResult(sucAttrLength, data, rulesSucAttr,
            decAreaDecSuc, finalVal));

      ruleData.append("conf = ");
      ruleData.append(resultConf);
```

```
            ruleData.append(" supp = ");
            ruleData.append(resultSupp);
            ruleData.append(" lift = ");
            ruleData.append(resultLift);
            ruleData.append("\n");

            result.addRule(ruleData.toString());
        }

        return result;
    }

    /**
     * Evaluate the value of the records against the decomposed area and define
     * if the attributes combination is valid or not.
     * @param attrLength amount of attributes elements to evaluate.
     * @param records matrix with the records.
     * @param recordPos actual record position to evaluate (row).
     * @param attrList array of attributes to evaluate.
     * @param decompArea matrix with the result of the rule decomposition.
     * @param index actual rule position (antecedent or succedent).
     * @return true if the attributes are into the range of the decomposed area,
     * false if not.
     */
    private static boolean testAttributes(int attrLength, int[][] records,
            int recordPos, int[] attrList, int[][] decompArea, int index) {
        int testNumber = 0;
        boolean validAttr = false;

        for (int count = 1; count <= attrLength; count++) {
            final int recordValue = records[recordPos][attrList[count - 1]];
            final int antValueIni = decompArea[index][2 * (count - 1)];
            final int antValueFin = decompArea[index][(2 * (count - 1)) + 1];

            if ((recordValue >= antValueIni) && (recordValue <= antValueFin)) {
                testNumber++;
            }
        }

        if (testNumber == attrLength) {
            validAttr = true;
        }

        return validAttr;
    }

    /**
```

```
 * Generate the valid rule structure.
 * @param attrLength quantity of elements in the attributes.
 * @param data the data input elements.
 * @param attrList list of the attributes elements.
 * @param decompArea matrix with the area decomposition values.
 * @param posVal position of the element in the decomposition area.
 * @return the valid rule as result.
 */
private static String createResult(int attrLength, dataInput data,
        int[] attrList, int[][] decompArea, int posVal) {
    StringBuilder ruleData = new StringBuilder(400);

    for (int index = 1; index <= attrLength; index++) {
        final String attrName =
                data.getAttributesElement(attrList[index - 1]);
        final int initialPos = (int) Math.ceil(
                decompArea[posVal][2 * (index - 1)]);
        final int finalPos = (int) Math.floor(
                decompArea[posVal][(2 * (index - 1)) + 1]);

        if (initialPos != finalPos) {
            ruleData.append(attrName);
            ruleData.append(" = ");
            ruleData.append(initialPos);
            ruleData.append("..");
            ruleData.append(finalPos);
            ruleData.append(' ');
        } else {
            ruleData.append(attrName);
            ruleData.append(" = ");
            ruleData.append(initialPos);
            ruleData.append(' ');
        }
    }

    return ruleData.toString();
}

/**
 * Get the support for the rule being analyze.
 * @param AntCompSucComp number of records that satisfy with the antecedent
 * and succedent of the rule.
 * @param recordsNumber quantity of records.
 * @return the support value for the rule.
 */
private static float getRuleSupport(int AntCompSucComp, int recordsNumber) {
    float support = 0;
```

```
    final float records = (float) recordsNumber;
    final float attrComp = (float) AntCompSucComp;

    if (records != 0) {
        support = attrComp / records;
    }

    return support;
}

/**
 * Get the confidence for the rule being analyze.
 * @param AntCompSucComp number of records that satisfy with the antecedent
 * and succedent of the rule.
 * @param AntCompSucUncomp number of records that satisfy with the
 * antecedent of the rule.
 * @return the confidence value for the rule.
 */
private static float getRuleConfidence(int AntCompSucComp,
        int AntCompSucUncomp) {
    float confidence = 0;
    final float antCsucC = (float) AntCompSucComp;
    final float antCsucU = (float) AntCompSucUncomp;
    final float addAntSuc = antCsucC + antCsucU;
    if (addAntSuc != 0) {
        confidence = antCsucC / addAntSuc;
    }

    return confidence;
}

/**
 * Get the lift for the rule being analyze.
 * @param AntCompSucComp number of records that satisfy with the antecedent
 * and succedent of the rule.
 * @param AntCompSucUncomp number of records that satisfy with the
 * antecedent of the rule.
 * @param AntUncompSucComp number of records that satisfy with the succedent
 * of the rule.
 * @param recordsNumber quantity of records.
 * @return the lift value for the rule.
 */
private static float getRuleLift(int AntCompSucComp, int AntCompSucUncomp,
        int AntUncompSucComp, int recordsNumber) {
    float lift = 0;
    final float records = (float) recordsNumber;
    final float antCsucC = (float) AntCompSucComp;
```

```
        final float antCsucU = (float) AntCompSucUncomp;
        final float antUsucC = (float) AntUncompSucComp;
        final float addAntCCSucCU = antCsucC + antCsucU;
        final float addAntCCSucUC = antCsucC + antUsucC;

        if ((addAntCCSucCU != 0) && (records != 0)) {
            lift = (antCsucC / addAntCCSucCU) / (addAntCCSucUC / records);
        }

        return lift;
    }

    /**
     * Short the value of the rules verification elements (confidence, support,
     * lift) to show them in a user-friendly way.
     * @param number the number to be shorten.
     * @return the number shortened.
     */
    private static String shortenValue(float number) {
        if (String.valueOf(number).length() > 4) {
            return String.valueOf(number).substring(0, 4);
        } else {
            return String.valueOf(number);
        }
    }
}
```

2.4.3. quargObjects Package

2.4.3.1. dataInput.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargObjects;

import commonFunctions.matrices;

/**
 * Represents the object that contains the data input values.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
```

```
public final class dataInput {

    /** List of attributes (columns). */
    private final String[] attributes;

    /** Matrix with all the input records (rows). */
    private int[][] records;

    /** Minimum number of antecedent combinations. */
    private final int antecedentMin;

    /** Maximum number of antecedent combinations. */
    private final int antecedentMax;

    /** Minimum number of succedent combinations. */
    private final int succedentMin;

    /** Maximum number of succedent combinations. */
    private final int succedentMax;

    /** List of attributes in use as antecedent. */
    private int[] antecedentAttributes;

    /** List of attributes in use as succedent. */
    private int[] succedentAttributes;

    /** Confidence. */
    private final float conf;

    /** Lift. */
    private final float lift;

    /** Support. */
    private final float supp;

    /** Quantity of attributes. */
    private final int attributesNumber;

    /** Quantity of records. */
    private final int recordsNumber;

    /**
     * Constructor.
     * @param attr Attributes separated by ';'.
     * @param rec Data records separated by ',' (rows) and ';' (columns).
     * @param antMin Antecedent minimum number of attributes combinations.
     * @param antMax Antecedent maximum number of attributes combinations.
```

```
 * @param sucMin Succedent minimum number of attributes combinations.
 * @param sucMax Succedent maximum number of attributes combinations.
 * @param antAttr List of the attributes that conforms the antecedent
 * (positions in array).
 * @param sucAttr List of the attributes that conforms the succedent
 * (positions in array).
 * @param c Confidence value.
 * @param l Lift value.
 * @param s Support value.
 */
public dataInput(String attr, String rec, int antMin, int antMax,
        int sucMin, int sucMax, String antAttr, String sucAttr,
        float c, float l, float s) {

    /* Attributes titles. */
    attributes = attr.split(";");

    /* Attributes quantity. */
    attributesNumber = attributes.length;

    /* Records data. */
    records = matrices.csvToIntMatrix(rec, attributesNumber);

    /* Records quantity. */
    recordsNumber = rec.split(",").length;

    /* Antecedent minimum combination. */
    antecedentMin = antMin;

    /* Antecedent maximum combination. */
    antecedentMax = antMax;

    /* Succedent minimum combination. */
    succedentMin = sucMin;

    /* Succedent maximum combination. */
    succedentMax = sucMax;

    /* Antecedent attributes. */
    final String[] antValues = antAttr.split(";");
    final int antValuesLength = antValues.length;
    antecedentAttributes = new int[antValuesLength];

    for (int i = 0; i < antValuesLength; i++) {
        antecedentAttributes[i] = Integer.valueOf(antValues[i]);
    }
```

```
    /* Succedent attributes. */
    final String[] sucValues = sucAttr.split(";");
    final int sucValuesLength = sucValues.length;
    succedentAttributes = new int[sucValuesLength];

    for (int i = 0; i < sucValuesLength; i++) {
       succedentAttributes[i] = Integer.valueOf(sucValues[i]);
    }

    /* Confidence. */
    conf = c;

    /* Lift. */
    lift = l;

    /* Support. */
    supp = s;
}

/**
 * Get the name of an attribute.
 * @param index the attribute array index position.
 * @return the attribute name.
 */
public final String getAttributesElement(int index) {
    return attributes[index];
}

/**
 * Get the matrix with the records.
 * @return a matrix with the records values.
 */
public final int[][] getRecords() {
    return records;
}

/**
 * Get the antecedent minimum combination value.
 * @return the minimum combination value for the antecedent.
 */
public final int getAntecedentMin() {
    return antecedentMin;
}

/**
 * Get the antecedent maximum combination value.
 * @return the maximum combination value for the antecedent.
```

```
 */
public final int getAntecedentMax() {
   return antecedentMax;
}

/**
 * Get the succedent minimum combination value.
 * @return the minimum combination value for the succedent.
 */
public final int getSuccedentMin() {
   return succedentMin;
}

/**
 * Get the succedent maximum combination value.
 * @return the maximum combination value for the succedent.
 */
public final int getSuccedentMax() {
   return succedentMax;
}

/**
 * Get the list of antecedent attributes id.
 * @return an array with the id values.
 */
public final int[] getAntecedentAttributes() {
   return antecedentAttributes;
}

/**
 * Get the list of succedent attributes id.
 * @return an array with the id values.
 */
public final int[] getSuccedentAttributes() {
   return succedentAttributes;
}

/**
 * Get the value of the confidence.
 * @return confidence value.
 */
public final float getConf() {
   return conf;
}

/**
 * Get the value of the lift.
```

```
     * @return lift value.
     */
    public final float getLift() {
        return lift;
    }

    /**
     * Get the value of the support.
     * @return support value.
     */
    public final float getSupp() {
        return supp;
    }

    /**
     * Get the quantity of attributes.
     * @return the quantity value.
     */
    public final int getAttributesNumber() {
        return attributesNumber;
    }

    /**
     * Get the quantity of records.
     * @return the quantity value.
     */
    public final int getRecordsNumber() {
        return recordsNumber;
    }

    /**
     * Generate an array with the column of records needed.
     * @param index position (column) of the records.
     * @return an array with the column of records.
     */
    public final int[] getRecordColumn(int index) {
        return matrices.getColumn(records, index, recordsNumber);
    }
}
```

2.4.3.2. associationRules.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
```

```
 */
package quargObjects;

import java.util.Arrays;
import commonFunctions.arrays;

/**
 * Represents an object for each of the rules that are going to be evaluated.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class associationRules {

   /** Id (positions) of the attributes that conform the antecedent. */
   private final int[] antAttributes;

   /** Id (positions) of the attributes that conform the succedent. */
   private final int[] sucAttributes;

   /** List of the elements (records) that conform the antecedent. */
   private final int[] antecedent;

   /** List of the elements (records) that conform the succedent. */
   private final int[] succedent;

   /** Next element of the list. */
   private associationRules next;

   /**
    * Constructor.
    * @param ant antecedent records.
    * @param suc succedent records.
    * @param antCols antecedent list of id.
    * @param sucCols succedent list of id.
    */
   public associationRules(int[] ant, int[] suc, int[] antCols,
         int[] sucCols) {

      /* Antecedent records combination. */
      antecedent = ant;

      /* Succedent records combination. */
      succedent = suc;

      /* Antecedent attributes array. */
      antAttributes = Arrays.copyOf(antCols, antCols.length);
```

```
    /* Succedent attributes array. */
    sucAttributes = Arrays.copyOf(sucCols, sucCols.length);

    /* Pointer to the next element of the list. */
    next = null;
}

/**
 * Get the maximum value from the antecedent records.
 * @return the maximum value.
 */
public final int getMaxAntecedentValue() {
    return arrays.getMaxValue(antecedent);
}

/**
 * Get the maximum value from the succedent records.
 * @return the maximum value.
 */
public final int getMaxSuccedentValue() {
    return arrays.getMaxValue(succedent);
}

/**
 * Set the value of the next list element.
 * @param n next association rule.
 */
public final void setNext(associationRules n) {
    next = n;
}

/**
 * Select a value from the antecedent records.
 * @param index the position in the array.
 * @return the antecedent record value.
 */
public final int getAntecedentElement(int index) {
    return antecedent[index];
}

/**
 * Select a value from the succedent records.
 * @param index the position in the array.
 * @return the succedent record value.
 */
public final int getSuccedentElement(int index) {
```

```
    return succedent[index];
}

/**
 * Select the next element in the list.
 * @return the next rule.
 */
public final associationRules getNext() {
    return next;
}

/**
 * Select the length of the antecedent attributes id list.
 * @return the length of the antecedent list.
 */
public final int getAntAttributesLength() {
    return antAttributes.length;
}

/**
 * Select the length of the succedent attributes id list.
 * @return the length of the succedent list.
 */
public final int getSucAttributesLength() {
    return sucAttributes.length;
}

/**
 * Get the list with the antecedent attributes id.
 * @return an array with the id.
 */
public final int[] getAntAttributes() {
    return antAttributes;
}

/**
 * Get the list with the succedent attributes id.
 * @return an array with the id.
 */
public final int[] getSucAttributes() {
    return sucAttributes;
}

/**
 * Get the array with the list of added records for the antecedent.
 * @return array with the added records.
 */
```

```
    public final int[] getAntecedent() {
        return antecedent;
    }

    /**
     * Get the array with the list of added records for the succedent.
     * @return array with the added records.
     */
    public final int[] getSuccedent() {
        return succedent;
    }
}
```

2.4.3.3. interestArea.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargObjects;

/**
 * Represents the object that contains the interest area values for a
 * certain rule.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class interestArea {

    /** Squares of coordinates (rows in coordinates matrix). */
    private final int squaresNum;

    /** The coordinates matrix of the rule's elements. */
    private final int[][] coordinates;

    /**
     * Constructor.
     * @param sNum the number of squares of coordinates.
     * @param coord the coordinates matrix.
     */
    public interestArea(int sNum, int[][] coord) {
        squaresNum = sNum;
        coordinates = coord;
    }
```

```
    /**
     * Select the value of squares.
     * @return the squares value.
     */
    public final int getSquaresNum() {
        return squaresNum;
    }

    /**
     * Select a specific coordinate value.
     * @param indexA index on the rows.
     * @param indexB index on the columns.
     * @return a coordinate value.
     */
    public final int getCoordinatesElement(int indexA, int indexB) {
        return coordinates[indexA][indexB];
    }
}
```

2.4.3.4. decomposedArea.java

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargObjects;

/**
 * Represents the object that contains the decomposed rule values.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
 */
public final class decomposedArea {

    /** Matrix with the values of the antecedent decomposition. */
    private int[][] decompAntecedent;

    /** Matrix with the values of the succedent decomposition. */
    private int[][] decompSuccedent;

    /** Value of the number of antecedent decompositions. */
    private int decompAntNumber;
```

```
/** Value of the number of succedent decompositions. */
private int decompSucNumber;

/**
 * Constructor.
 */
public decomposedArea() {
   decompAntecedent = null;
   decompSuccedent = null;
   decompAntNumber = 0;
   decompSucNumber = 0;
}

/**
 * Get the matrix of the antecedent decomposition.
 * @return the decomposed antecedent matrix.
 */
public final int[][] getDecompAntecedent() {
   return decompAntecedent;
}

/**
 * Get the matrix of the succedent decomposition.
 * @return the decomposed succedent matrix.
 */
public final int[][] getDecompSuccedent() {
   return decompSuccedent;
}

/**
 * Get the value of decomposed areas for the antecedent.
 * @return the quantity of decomposed areas.
 */
public final int getDecompAntNumber() {
   return decompAntNumber;
}

/**
 * Get the value of decomposed areas for the succedent.
 * @return the quantity of decomposed areas.
 */
public final int getDecompSucNumber() {
   return decompSucNumber;
}

/**
 * Increment the value of the antecedent decomposition number (rows in
```

```
     * the matrix).
     */
    public final void incrementAntNumber() {
        decompAntNumber++;
    }

    /**
     * Increment the value of the succedent decomposition number (rows in
     * the matrix).
     */
    public final void incrementSucNumber() {
        decompSucNumber++;
    }

    /**
     * Set the matrix with the decomposed antecedent values.
     * @param value the matrix with the decomposed values.
     */
    public final void setDecompAntecedent(int[][] value) {
        decompAntecedent = value;
    }

    /**
     * Set the matrix with the decomposed succedent values.
     * @param value the matrix with the decomposed values.
     */
    public final void setDecompSuccedent(int[][] value) {
        decompSuccedent = value;
    }
}
```

2.4.3.5. analyzedRule.java Class

```
/*
 * IDA Research Lab.
 *
 * This software is open to the community for future implementations and
 * modifications.
 */
package quargObjects;

/**
 * Represents the object that contains the analyzed rule data, this create a
 * sub-rule or a set of it depending on the interest areas analyzed.
 *
 * @version 1.0 Nov 2011
 * @author Adalberto Cubillo
```

```
 */
public final class analyzedRule {

    /** Text with the sub-rule(s) selection (depending in the interest area
     * analysis). */
    private String ruleResult;

    /** Number of sub-rules selected. */
    private int rulesNumber;

    /** Number of sub-rules verified. */
    private int verificationsNumber;

    /**
     * Constructor.
     */
    public analyzedRule() {
        ruleResult = "";
        rulesNumber = 0;
        verificationsNumber = 0;
    }

    /**
     * Show the resulting sub-rule(s).
     * @return the string with the sub-rule(s) details.
     */
    public final String getRuleResult() {
        return ruleResult;
    }

    /**
     * Show the quantity of sub-rules selected.
     * @return the quantity.
     */
    public final int getRulesNumber() {
        return rulesNumber;
    }

    /**
     * Show the quantity of verifications.
     * @return the quantity.
     */
    public final int getVerificationsNumber() {
        return verificationsNumber;
    }

    /**
```

```
     * Increment the value of the number of valid rules.
     */
    public final void incrementRulesNumber() {
        rulesNumber++;
    }

    /**
     * Increment the value of the number of rules evaluated.
     */
    public final void incrementVerificationsNumber() {
        verificationsNumber++;
    }

    /**
     * Add a rule to the result.
     * @param rule the rule text to be added.
     */
    public final void addRule(String rule) {
        StringBuilder buffer = new StringBuilder(800);
        buffer.append(ruleResult);
        buffer.append(rule);
        ruleResult = buffer.toString();
    }
}
```

3. Experimental Data and Test Results

The selected data for the runtime comparison between the experimental scripts and the web application was a sample of 6 attributes and 977 records taken from the real-life database STULONG, which is a study of the risk factors of atherosclerosis in middle aged men [2].

The comparison shows the following results:

Table 106. Results with one attribute combinations, for the comparison between scripts and application.

Executions	QUARG algorithm	Experimental scripts
1	12	432
2	11	439
3	10	424
4	12	483
5	14	425
6	11	435
7	11	456
8	14	438
9	14	449
10	12	440
Average	12.1	442.1

Note: the results are given in milliseconds.

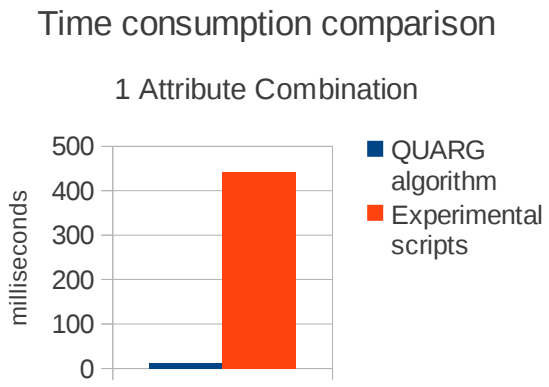

Figure 11. Graphical display of the results with one attribute combinations, for the comparison between scripts and application.

Table 107. Results with two attribute combinations, for the comparison between scripts and application.

Executions	QUARG algorithm	Experimental scripts
1	862	12173
2	857	12290
3	833	12207
4	895	12959
5	881	12591
6	888	12646
7	867	11928
8	887	11881
9	838	11041
10	876	11029
Average	868.4	12074.5

Note: the results are given in milliseconds.

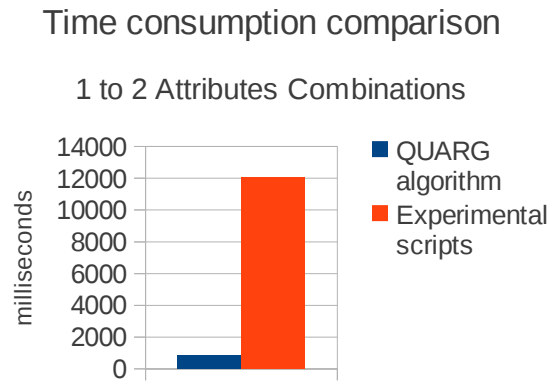

Figure 12. Graphical display of the results with two attribute combinations, for the comparison between scripts and application.

Table 108. Results with three attribute combinations, for the comparison between scripts and application.

Executions	QUARG algorithm	Experimental scripts
1	23053	74494
2	24694	69765
3	23963	59072
4	22896	69127
5	22696	81541
6	24135	75146
7	24320	79896
8	22878	66515
9	20729	61684
10	21939	82113
Average	23130.3	71935.3

Note: the results are given in milliseconds.

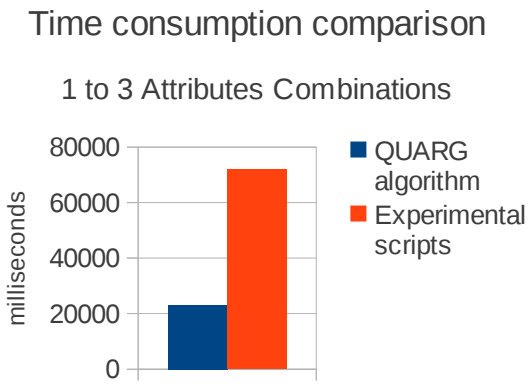

Figure 13. Graphical display of the results with three attribute combinations, for the comparison between scripts and application.

Additional to the comparison between the experimental scripts and the web application, were analyzed the runtime between the two implementations of rules mining algorithms, the genetic algorithm and the full search one, giving the following results:

Table 109. Results with one attribute combinations, for the comparison between the rules mining algorithms.

Executions	Genetic algorithm	Full search
1	12	45
2	11	45
3	10	66
4	12	52
5	14	44
6	11	52
7	11	48
8	14	47
9	14	47
10	12	51
Average	12.1	49.7

Note: the results are given in milliseconds.

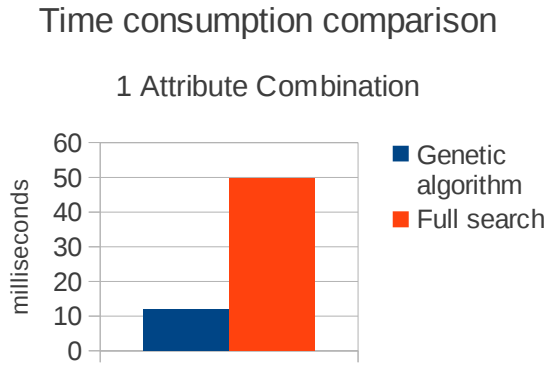

Figure 14. Graphical display of the results with one attribute combinations, for the comparison between the rules mining algorithms.

Table 110. Results with two attribute combinations, for the comparison between the rules mining algorithms.

Executions	Genetic algorithm	Full search
1	862	3183
2	857	3236
3	833	3252
4	895	3182
5	881	3202
6	888	3226
7	867	3227
8	887	3226
9	838	3274
10	876	3222
Average	868.4	3223

Note: the results are given in milliseconds.

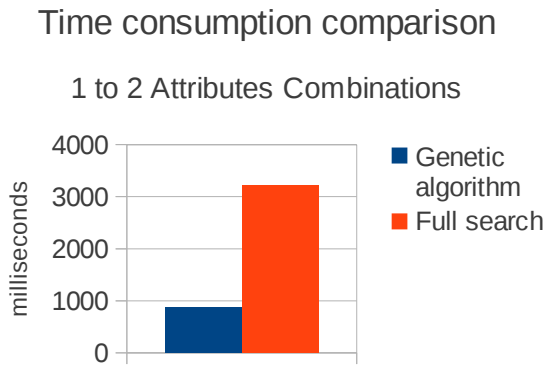

Figure 15. Graphical display of the results with two attribute combinations, for the comparison between the rules mining algorithms.

Table 111. Results with three attribute combinations, for the comparison between the rules mining algorithms.

Executions	Genetic algorithm	Full search
1	23053	47456
2	24694	47888
3	23963	47750
4	22896	47822
5	22696	47872
6	24135	47809
7	24320	47782
8	22878	47724
9	20729	47884
10	21939	47856
Average	23130.3	47784.3

Note: the results are given in milliseconds.

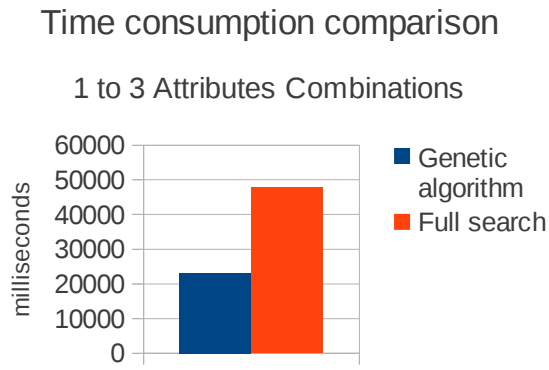

Figure 16. Graphical display of the results with three attribute combinations, for the comparison between the rules mining algorithms.

4. Conclusions

As can be seen on the results, the execution time of the QUARGsoft application has improved more than 50%, compared with the experimental scripts. The project accomplish its main objective of developing a more efficient and standardized application for the implementation of the QUARG algorithms. But taking into account, that this application does not meet the enough optimization to execute data of large-scale.

Also, the project accomplish with other objectives as generating an extensive source code documentation and a detailed user guide which explain the process of installing, configure an to use the application, which will help in a future reimplementation of modification of the project (see attachment).

Within the documents and products submitted are:

- Source code of the web application.
- Source code of the web service.
- Source code of the command application.
- On-line server with the application running.
- Subversion server with a repository of all the documents and products, and accessible on-line.
- Javadoc of the functionalities and methods.
- Installer file of QUARGsoft Java web server (.war).
- Compressed Java file (.jar) for executing the application from terminal or command line.
- User guide (Installation, Configuration and Usage of the application).

The experience gained during the internship includes knowledge about genetics algorithms and data mining techniques, a better use of Java conventions and internal libraries, a more comprehensive knowledge about web development tools (JQuery, PHP, Ajax, Web Services, CSS), Unix systems employment (Linux), and knowledge about tools related to project's planning.

5. Future Work

As possibilities for future projects related to the QUARGsoft application are:

- The possibility of adding more rules mining functionalities to the application.
- Take the actual software functionalities and implement them in more spread platforms or tool (E.g. Weka Data Mining Software).
- Implement the application in distributed systems to analyze the behavior and runtime of the algorithms.
- Propose changes on the theoretical foundations of the application to improve the functionalities.

6. Bibliography

1. Czech Technical University in Prague, “University History”, 2007, http://www.cvut.cz/en/history.

2. EuroMISE, “Project STULONG,” 2007, http://euromise.vse.cz/stulong-en/.

3. F. Karel, “Quantitative association rules mining,” Ph. D. thesis, Czech Technical University in Prague, Prague, Czech Republic, 2009.

4. IDA Research Lab, “About Us”, s.f, http://ida.felk.cvut.cz/ida/about.

Costa Rica Institute of Technology
School of Computer Engineering

Intelligent Data Analysis Research Lab
Czech Technical University in Prague

DEVELOPMENT OF A SOFTWARE APPLICATION FOR THE EXECUTION OF THE QUANTITATIVE ASSOCIATION RULES MINING ALGORITHM (QUARG)

User Guide

Adalberto Cubillo Rojas
ID: 200838881

Prague
January, 2012

Table of Contents

List of Figures

1. Introduction

The Quantitative Association Rules Mining Algorithm (QUARG) was developed for searching relationships within attributes in large databases of information. This algorithm unlike the common searching algorithms use data preprocessing and genetic algorithms to predict which are the best rules and improve the performance in the rules mining runtime.

This algorithm was implemented in the QUARGsoft web application and in a Java compress executable file (.jar) to allow the users run the application on-line or locally. For more details about the QUARG algorithm and QUARGsoft refer to the final report of the project.

The main objective of this document is to explain how to install, configure and use the web application and the command line application for the QUARG algorithm, in both Unix-based systems as Windows systems.

2. Installation guide

The software selected to host and run the QUARGsoft application are:

- **Apache Server**: to host the website and the graphical user interface, this server was choose because its widespread, the ease of installation and configuration, and the large number of complements that have been developed to this server (E.g. PHP complement, SOAP connectivity, …).

- **PHP**: as the controller layer between the website and the web server, which increment the security and allows a better design of the application.

- **Java**: one of the most widespread programming languages, with high performance, object-oriented, and a large number of functionalities.

- **GlassFish Server**: to host the Java web server, and the open source edition was choose because the ease of installation, the high level of customization, and the high performance.

2.1. Apache Server 2.2

Unix-like systems:

- Download Apache HTTP server from http://httpd.apache.org/download.cgi or using the following command in the terminal:

$ lynx http://httpd.apache.org/download.cgi

- Extract the source using the command:

$ gzip -d httpd-NN.tar.gz
$tar xvf httpd-NN.tar

NN must be replaced with the current version number, don't forget to cd into the new directory containing the source code for proceeding with compiling the server.

$ cd httpd-NN

- Configure the Apache HTTPd source tree for your particular platfrom and personal requirements.

$ ./configure --prefix=PREFIX

To configure the source tree using the default options simply type *./configure*. Otherwise, you must replace PREFIX with the filesystem path under which the server should be installed (Default to */usr/local/apache2*).

- Build the Apache HTTPd package running the command:

$ make

- And install Apache with the command:

$ make install

- To start/stop the server execute the following commands in the terminal:

$ /usr/local/apache2/bin/apachetl start
$ /usr/local/apache2/bin/apachetl stop

- Finally, copy the content of the QUARGsoft_website.zip file in the directory */var/www/html*.

Windows systems (Windows 2000 or later):

- Download Apache HTTP server from http://httpd.apache.org/download.cgi.
- Run the Apache .msi file.
- The installation will ask for the following:
    - **Network Domain**: Enter the DNS domain in which the server will be registered.
    - **Server name**: Your server's full DNS name.

- **Administrator's email address**: Enter the server administrator's or webmaster's email address.
- **For whom to install Apache**: Select for All Users, on Port 80, as Service.
- **The installation type**: Select Typical for everything except the source code and libraries for module development.
- **Where to install**: The default path is *"C:\Program Files\Apache Software Foundation\Apache2.2"*.

- To start/stop the server execute the following commands in the command window:

  httpd.exe start
  httpd.exe stop

  Should be in the directory where Apache is installed.

- Finally, copy the content of the QUARGsoft_website.zip file in the directory *"C:\Program Files\Apache Software Foundation\Apache2.2\htdocs"*.

2.2. PHP 5.3.9

Unix-like systems:

- Download PHP from the http://www.php.net/downloads.php.

- Unpack the source using the command:

  $ gunzip php-NN.tar.gz
  $tar -xf php-NN.tar

  NN must be replaced with the current version number, don't forget to cd into the new directory containing the source code for proceeding with compiling the server.

  $ cd php-NN

- Now, configure and build PHP

  $ ./configure --with-apxs2=/usr/local/apache2/bin/apxs --enable-soap

  Take in consideration that sometimes the folder apxs change to apxs2 in Apache.

- Build the PHP package running the command:

  $ make

- Install PHP with the command:

$ make install

- And setup your php.ini file:

$ cp php.ini-production /usr/local/lib/php.ini

- Finally, restart Apache server.

Windows systems:

- Download the Windows PHP installer using the Wix Toolkit from http://wix.sourceforge.net/.
- Run the MSI installer and follow the instructions provided by the installation wizard.
    - You will be prompted to select the Web Server you wish to configure first (Apache).
    - After that you can select which features and extensions you wish to install and enable, in this case you have to select SOAP feature.
- Finally, restart Apache server.

2.3 Java JDK 7

Unix-like systems:

- Download the Java Development Kit archive binary file (.tar.gz) from http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- Move the file to the directory on which you want to install the JDK.
- Unpack the tarball and install the JDK:

$ tar zxvf jdk-7u<version>-linux-i586.tar.gz

- Delete the .tar.gz file to save disk space.

Windows systems:

- Download the Java Development Kit self-installing executable file from http://www.oracle.com/technetwork/java/javase/downloads/index.html.
- Run the file *jdk-7<version>-windows-i586-i.exe*, and follow the installation instructions.

2.4 GlassFish Server

Unix-like systems:

- Download the self-extracting file (.sh) from http://glassfish.java.net/downloads/3.1.1-final.html
- At the terminal type the follow command to start the installation:

$ sh ./self-extracting-file-name.sh

- Choose the installation type as Typical installation.
- Specify where to install the GlassFish Server (Default: *user-home-directory/glassfish3*).
- Press the button install.
- The installer will create the default domain “domain1” copy the QUARGsoft_webservice.war into your user directory (or a similar).
- Start the GlassFish server with the following command:

$ as-install/bin/asadmin start-domain

In case of needed to be stopped:

$ as-install/bin/asadmin stop-domain

- And finally, deploy the QUARGsoft application with the following command:

$ as-install/bin/asadmin deploy user-directory/ QUARGsoft_webservice.war

Windows systems:

- Download the executable file (.exe) from http://glassfish.java.net/downloads/3.1.1-final.html
- Double click the executable .exe installation file to start the installation process.
- Choose the installation type as Typical installation.
- Specify where to install the GlassFish Server (Default: *C:\glassfish3*).
- Press the button install.
- The installer will create the default domain “domain1” copy the QUARGsoft_webservice.war into your user directory (or a similar).

- Start the GlassFish server with the following command:

  $ as-install/bin/asadmin start-domain

  In case of needed to be stopped:

  $ as-install/bin/asadmin stop-domain

- And finally, deploy the QUARGsoft application with the following command:

  $ as-install/bin/asadmin deploy user-directory/ QUARGsoft_webservice.war

3. Configuration guide

The following steps are for configuring the Apache server + PHP and GlassFish server to execute the QUARGsoft website and webservice, in case of any other extra modification refer to the respective installation or user guide for each platform.

3.1. Apache Server + PHP configuration

- Edit your Apache httpd.conf file to load the PHP module, adding the follow line:

  LoadModule php5_module modules/libphp5.so

  The installation may have already added this, but be sure to check.

- Tell Apache to parse certain extensions as PHP addind the follwing code lines to the httpd.conf file:

```
<FilesMatch "\.ph(p[2-6]?|tml)$">
        SetHandler application/x-httpd-php
</FilesMatch>

<FilesMatch "\.phps$">
        SetHandler application/x-httpd-php-source
</FilesMatch>
```

- Open the php.ini file and modify the following lines:

  - Set the **default_socket_timeout** = 3600.
  - Set the **file_uploads** = On.
  - Set the **upload_max_filesize** = 5M.

- Set the **max_execution_time** = 3600.

These variables can be changed to other numeric values depending on the complexity of the data to analyze.

- Restart the Apache server.

3.2. GlassFish configuration

- Access to the GlassFish Administration Console using the address http://localhost:4848 or http://<webservice-url>:4848.

- After that, select in the **Common Tasks** tree the option **Configuration**.

- In the displayed list select the option **Web Container** and select the tab **Session Properties**.
- In there modify the **Session Timeout** variable set it to 3600 seconds, and press the button **Save**.

- Return to the **Configuration** option and select **EJB Container**.

- In the tab **EJB Settings**, modify the **Pool Idle Timeout** and the **Cache Idle Timeout** variables to 3600 seconds, and press the button **Save**.

- Select the tab MDB Settings and change the Idle Timeout to 3600 seconds, and press the button **Save**.

- Return to the **Configuration** option and select **Thread Pools**, and select the thread pool for the domain (First http-thread-pool, and after that thread-pool-1).

- In each one change the value to the variable **Idle Thread Timeout** to 3200 seconds, and press the button **Save**.

- Exit the GlassFish Administration Console and restart the GlassFish server.

4. Usage guide

4.1. QUARGsoft website application

The QUARGsoft web application is conformed by three sections: Data input and preprocessing, Analysis settings and Results. Below is the description of each section.

4.1.1. Data input and preprocessing

The first functionality in the application is the tab of **Data input and preprocessing**, which is in charge of receive the files with the attributes and records, and apply to them the discretization algorithms.

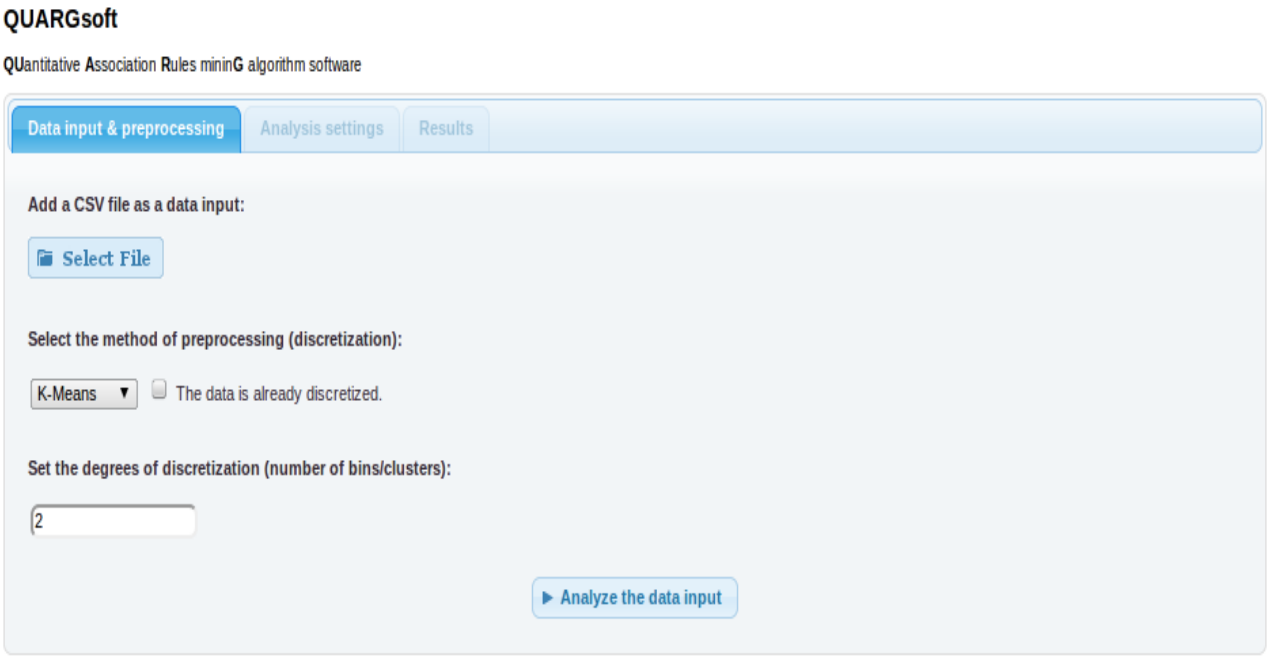

Figure 1. Screenshot of the Data input and preprocessing tab.

The components in this tab are:

- **Add a CSV file as data input**: there is the “Select File” button which allow the user to upload a file with the data input to the server.

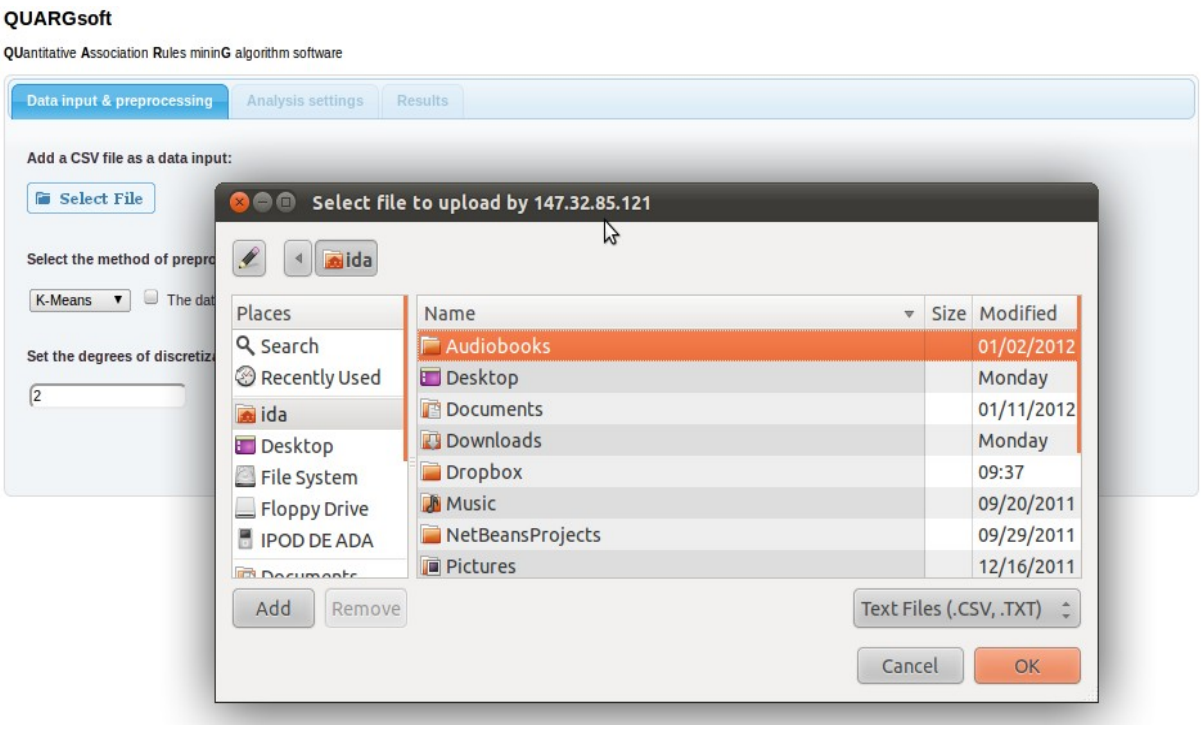

Figure 2. Example of the action when pressing the button of Select File.

The content of this file is in the format coma separated values (CSV) where the data is distributed by: the columns of attributes (separated by semicolon ';') and the rows of records (a record per line in the text), where the first line in the file represents the attributes names and the rest of the lines the records.

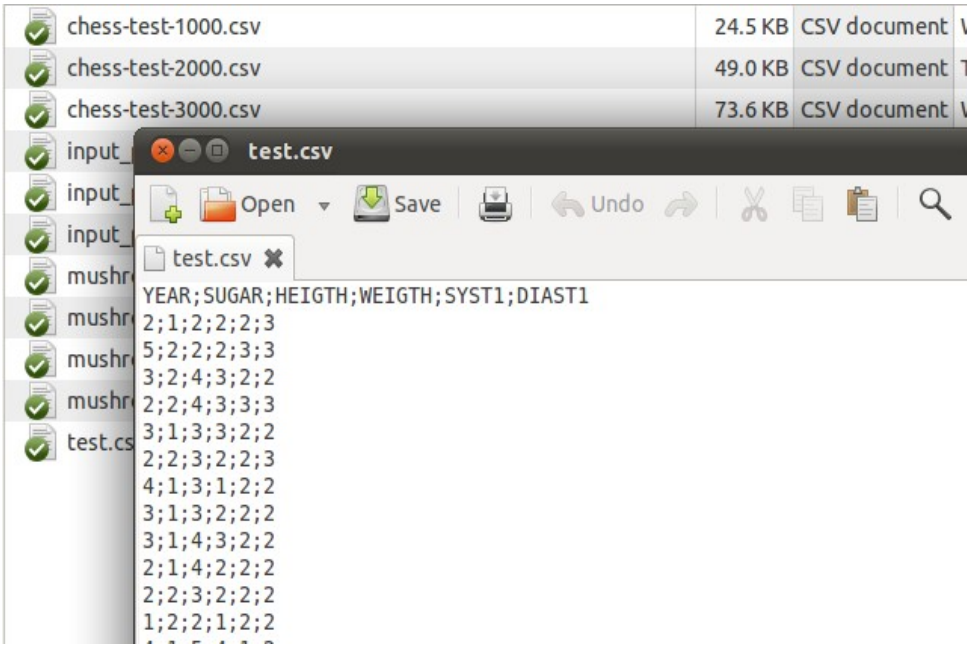

Figure 3. Example of a data input CSV file for the QUARGsoft application.

- **Select the method of preprocessing**: through the selector the user can choose between the three types of discretization methods applied in the program: K-Means, Equi-Depth and Equi-Width.

  If the input file is already preprocessed then the user only check the option “The data is already discretized” and skip this step of the application pressing the “Analyze the data input” button.

- **Set the degrees of discretization**: in this element of the form the user introduce the number of discretization bins or clusters, by default this value is set by 2 and only accept values higher or equal than 1.

After setting the previous information, just press the “Analyze the data input” button to execute the discretization process.

4.1.2. Analysis settings

After preprocessing the input data, the QUARGsoft application will fill the form in the **Analysis settings** tab, which handles the configuration of the data mining execution.

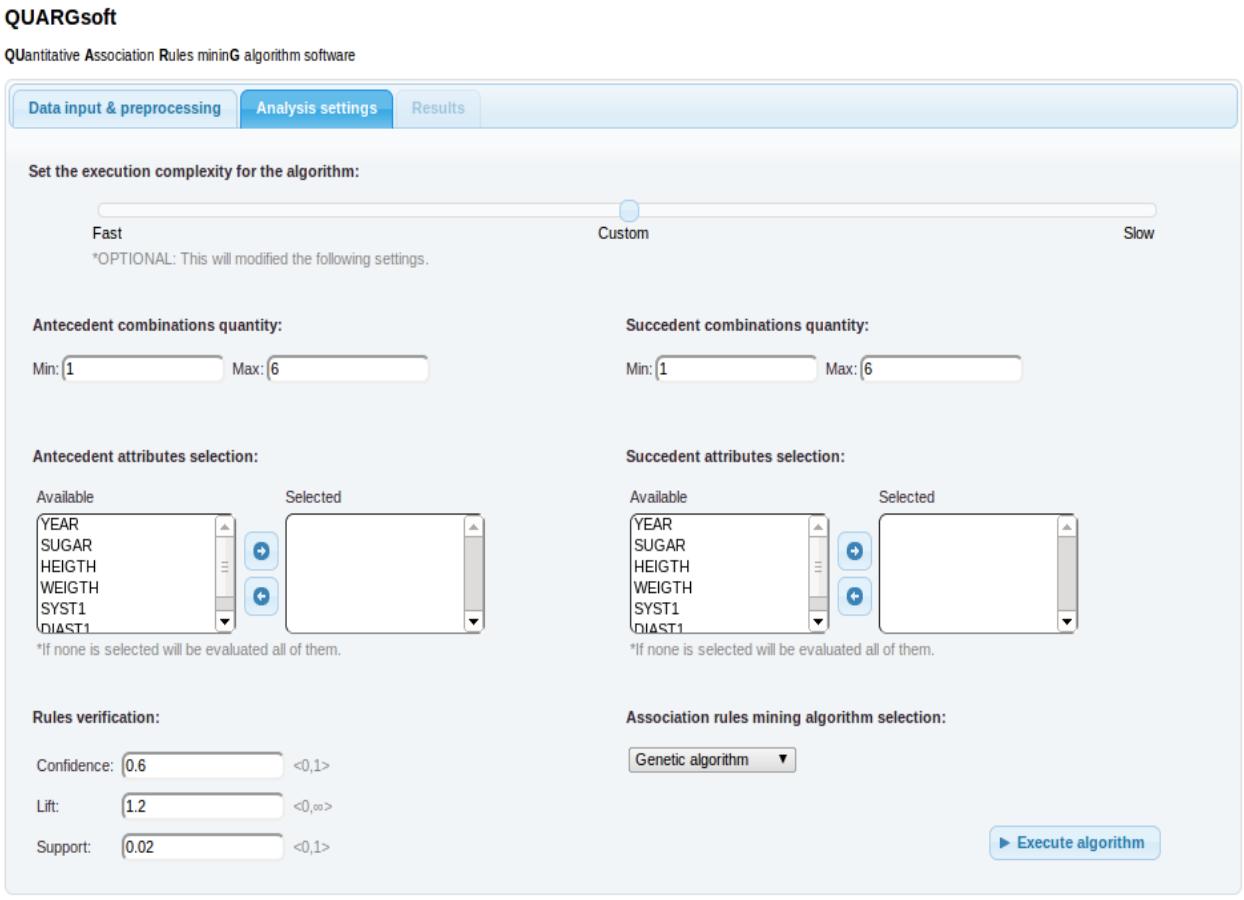

Figure 4. Screenshot of the Analysis settings tab.

The components in this form are:

- **Set the execution complexity for the algorithm**: the slider in this section let the user to choose the complexity of the data mining process, which affects directly the execution time of the analysis.

  Choosing the “Slow” option will execute all the possible combinations and set the rules verification elements to the lowest values, in contrast, choosing the “Fast” option will analyze only one attribute combination and set the rules verification elements with higher threshold values.

  The “Custom” option of the slider will set in case of applying any manual change to the rest elements of the form.

- **Antecedent combinations quantity**: this section allows the user to set the minimum and maximum amount of antecedent combinations that has to be generated during the data mining process.

  The maximum value cannot exceed the amount of chosen elements and cannot be less than the minimum value. And the minimum value has to be 1 or higher.

- **Succedent combinations quantity**: in the same way that the Antecedent combinations quantity, this section allows the user to set the minimum and maximum amount of succedent combinations that has to be generated during the data mining process. And applies the same restrictions as the Antecedent combinations quantity section.

- **Antecedent attributes selection**: allows the user to select which attributes form the antecedent elements in the rules, if none is selected then the application will execute the algorithm evaluating the antecedent with all the attributes.

- **Succedent attributes selection**: in the same way that the Antecedent attributes selection, this section allows the user to select which attributes form the succedent elements in the rules, and applies the same conditions than the other section.

- **Rules verification**: these elements of the Analysis settings establish the rule verification thresholds, divided in three: Confidence (values from 0 to 1), Lift (positive values) and Support (values from 0 to 1).

- **Association rules mining algorithm selection**: through the selector the user can choose between two types of rules mining techniques: Genetic algorithm or Full search algorithm, being the first one less demanding of time execution than the second one.

Once ready the analysis settings values, to continue with the process just press the “Execute algorithm” button to execute the data mining process.

4.1.3. Results

The last section of the application is the Results tab, which display the results of the rules mining algorithm grouped by rule, also shows how many rules possibilities were verified, how much were selected, and the execution time of the algorithm in milliseconds.

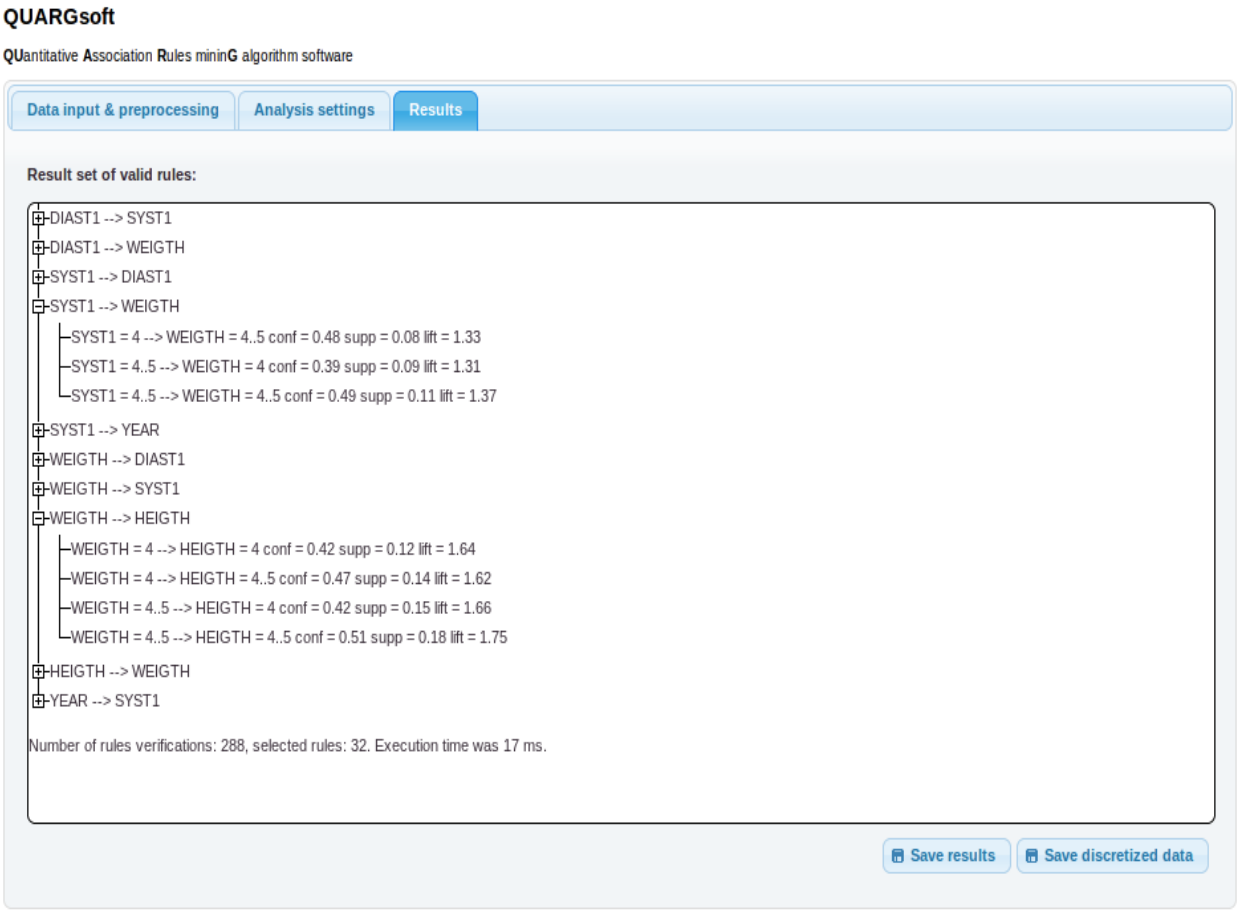

Figure 5. Screenshot of the Results tab.

This tab has the following components:

- **Result set of valid rules**: this element groups the valid rules by the general format and attributes of the rules, this is for a better understanding and organization of the results. And shows some general statistics about the algorithm execution (number of rules verified, number of rules selected, runtime).

- **Save results**: this button allows the user to save in a file the results of the rules mining algorithm execution, the file name will have the format:

<input-file-name>-results.txt

- **Save discretized data**: this button allows the user to save in a file the results of the preprocessing algorithm execution, the file name will have the format:

<input-file-name>-discretized.csv

4.2. QUARGsoft terminal application

Other way to execute the QUARGsoft application is using the QUARGsoft_terminal.jar version of it, which can be executed from the terminal with the "*java -jar QUARGsoft_terminal.jar*" command, besides this command is necessary the following parameters combinations.

4.2.1. Run the preprocessing algorithm

For executing the preprocessing method of the QUARGsoft application is necessary to used the keyword **discretize** as first parameter of the command, in addition you should add these other parameters in the following order:

- **Input file**: this parameter establish the name of the file with the data input to discretize, if the file is not in the same folder that the .jar file then is necessary to set the path to the file.

- **Number of bins/clusters**: the third parameter is the number of discretization bins or clusters, this value should be an integer number equal or greater than 1.

- **Algorithm type**: the last parameter is the keyword for the kind of preprocessing, which the options are **depth**, **width** or **kmeans**.

Example:

$ java -jar QUARGsoft_terminal.jar discretize /home/Documents/test.csv 5 width

The command will execute the discretization function with:

- the file test.csv.
- 5 discretization bins.
- the type method Equi-Width.

The result of the execution is a file in the same folder than the input file with the following format:

<input-file-name>-discretized.csv

4.2.2. Run the rules mining algorithm

The other functionality that can be execute from the terminal is the rules mining function, using the keyword **analyze** as the first parameter, in addition you should add these other parameters in the following order:

- **Input file**: this parameter establish the name of the file with the data input to analyze, if the file is not in the same folder that the .jar file then is necessary to set the path to the file.

- **Antecedent minimum combination**: this is the minimum antecedent attributes combination number, should be equal or greater than 1.

- **Antecedent maximum combination**: this is the maximum antecedent attributes combination number, should be equal or greater than 1, and cannot be less than the minimum value.

- **Succedent minimum combination**: this is the minimum succedent attributes combination number, should be equal or greater than 1.

- **Succedent maximum combination**: this is the maximum succedent attributes combination number, should be equal or greater than 1, and cannot be less than the minimum value.

- **Antecedent attributes selection**: the next parameter is the list with the indexes (according the order of the attributes names in the file) of attributes for the antecedent, with the following format: *'index1;index2;index3;...;indexN'*.

- **Succedent attributes selection**: the next parameter is the list with the indexes (according the order of the attributes names in the file) of attributes for the succedent, with the following format: *'index1;index2;index3;...;indexN'*.

- **Confidence**: the float value for the confidence on the rules verification, with values from 0 to 1.

- **Lift**: the float value for the lift on the rules verification, with only positive values.

- **Support**: the float value for the support on the rules verification, with values from 0 to 1.

- **Algorithm type**: the last parameter is the keyword for the kind of rules mining, which the options are **genetic** or **full**.

Example:

$ java -jar QUARGsoft_terminal.jar analyze /home/Documents/test.csv
1 2 2 3 '0;1;2' '0;1;4' 0.2 1 0.02 genetic

The command will execute the rules mining function with:

- the file test.csv.
- antecedent attributes combination from 1 (minimum) to 2 (maximum) attributes combinations.
- succedent attributes combinations from 2 (minimum) to 3 (maximum) attributes combinations.
- antecedent attributes selection with the attributes with the position 0, 1 and 2 in the file.

- succedent attributes selection with the attributes with the position 0, 1 and 4 in the file.
- confidence equal to 0.2.
- lift equal to 1
- support equal to 0.02.
- the type method Genetic Algorithm.

The result of the execution is a file in the same folder than the input file with the following format:

<input-file-name>-result.txt

Minutes #1 Approval of the
Internship Scopes and Schedule

Place: IDA Research Lab.
Date: October 3, 2011.

Present:
Jiří Kléma.
Adalberto Cubillo.

Agenda:
1. Definition of the internship project scopes.
2. Presentation of the internship project schedule.
3. Proposal of the programming language in which the project is going to be implemented.

Content:
1. Adalberto Cubillo defined the project scope, based on the internship project proposal, and Jiří Kléma read the proposal and accepted it.
2. The project schedule was introduced to Jiří Kléma, and is going to be send a copy to him for future analysis.
3. Java and C++ were proposed as the programming languages for the implementation of the project, leaving the decision of choose it to Adalberto Cubillo.

Jiří Kléma
Project Manager

Adalberto Cubillo
Intern

Minutes #2 Definition of the First Report Contents

Places: San José, Costa Rica,
Prague,Czech Republic.
Date: October 14, 2011.

Present:
Santiago Nuñez.
Adalberto Cubillo.

Agenda:

1. Definition of the first report contents.

Content:

a) Santiago proposed the following report contents and structure:

1. Short description of the university and the investigation laboratory.
2. Detailed description of the problem and project background.
3. Project description.
   a) Name of the project.
   b) Project description.
   c) General objective.
   d) Specific objectives.
   e) Expected products.
   f) Scopes and limitations.
   g) Methods and technologies.
   h) Initial schedule with estimated dates and activities dependencies.
   i) Gantt chart of the project.
4. Current advance and obtained results.

b) The proposed structure was accepted as the first report structure.

Santiago Nuñez
Project Advisor

Adalberto Cubillo
Intern

Minutes #3 Definition of the Second Report Contents

Places: San José, Costa Rica,
Prague,Czech Republic.
Date: December 7, 2011.

Present:

Santiago Nuñez.
Adalberto Cubillo.

Agenda:

1. Definition of the second report contents.

Content:

a) The following report contents and structure were proposed:

1. QUARG algorithm detailed description.
2. Application design.
   a) Classes diagram.
   b) Components diagram.
   c) Sequence diagram.
   d) Activities diagrams (Overall and specific diagram).
   e) Algorithm pseudo-code.
3. Source code.
4. Minutes and advance reports.

b) The proposed structure was accepted as the second report structure.

Santiago Nuñez
Project Advisor

Adalberto Cubillo
Intern

Minutes #4 Definition of the Final Report Contents

Places: San José, Costa Rica,
Prague,Czech Republic.
Date: January 12, 2012.

Present:
Santiago Nuñez.
Adalberto Cubillo.

Agenda:

1. Definition of the final report contents.

Content:

a) The following report contents and structure were proposed:

1. Executive summary.
2. Description of problem solved.
3. Description of the implemented solution.
4. Experimental data and test results.
5. Final diagrams, algorithms and source code.
6. Conclusions.
7. Future work.

b) The proposed structure was accepted as the final report structure.

Santiago Nuñez
Project Advisor

Adalberto Cubillo
Intern

Progress Report # 1

Name: Adalberto Cubillo Rojas.
Weeks: from 19 of September to 30 of September 2011.

Planned activities:

1. Project introduction and meeting with the professor in charge of the project.
2. Reading of the theoretical base of the project.
3. Experimental script analysis.
4. Adviser meeting.
5. Research and learning of Matlab.
6. Research of the best programming language for an optimal reimplementation of the experimental scripts.
7. Drafting of the detailed project definition.
8. Drafting and approval of the project scopes.
9. Drafting of the general schedule.

Realized activities as planned:

1. Project introduction and meeting with the professor in charge of the project.
2. Reading of the theoretical base of the project.
3. Experimental script analysis.
4. Research and learning of Matlab.
5. Research of the best programming language for an optimal reimplementation of the experimental scripts.
6. Drafting of the detailed project definition.
7. Drafting of the general schedule.

Realized activities no planned:

1. Installation of the development environment (Ubuntu), and the development tools (Matlab, Netbeans).
2. Debugging and testing of the experimental scripts.

Pending activities for the next 2 weeks:

1. Adviser meeting.
2. Drafting and approval of the project scopes.

Activities for the next 2 weeks:

1. Experimental script analysis, testing and debugging.
2. Investigation of the development language.
3. Meeting with the project manager for the approval of the development language.
4. Writing of the problem description and project context.
5. Creation of the domain model.

Progress Report # 2

Name: Adalberto Cubillo Rojas.
Weeks: from 03 of October to 14 of October 2011.

Planned activities:

1. Analysis of the actual experimental scripts.
2. Investigation of the development language.
3. Writing of the project context.
4. Writing of the problem description.
5. Detailed schedule delivery.
6. Adviser meeting.

Realized activities as planned:

1. Analysis of the actual experimental scripts.
2. Investigation of the development language.
3. Writing of the project context.
4. Detailed schedule delivery.

Realized activities no planned:

1. Drafting and approval of the project scopes.
2. Graphical user interface development.
3. System web service development.

Pending activities for the next 2 weeks:

1. Adviser meeting.
2. Writing of the problem description.

Activities for the next 2 weeks:

1. Definition of the use cases diagram and specification.
2. Risk analysis.
3. First report delivery.

Progress Report # 3

Name: Adalberto Cubillo Rojas.
Weeks: from 17 of October to 28 of October 2011.

Planned activities:

1. Class diagram definition.
2. Components diagram definition.
3. Application activities diagram definition.
4. Sequence diagram definition.
5. System state diagram definition.
6. Adviser meeting.
7. Gantt chart and detailed schedule definition.
8. Writing first report.

Realized activities as planned:

1. Class diagram definition.
2. Components diagram definition.
3. Application activities diagram definition.
4. Adviser meeting.
5. Gantt chart and detailed schedule definition.
6. Writing first report.

Realized activities no planned:

1. Functionalities development.
2. Functionalities documentation.

Pending activities for the next 2 weeks:

1. Sequence diagram definition.
2. System state diagram definition.

Activities for the next 2 weeks:

1. Functions pseudocode development.
2. Functionalities implementation.

Progress Report # 4

Name: Adalberto Cubillo Rojas.
Weeks: from 31 of October to 11 of November 2011.

Planned activities:

1. Application specific diagrams of activities definition.
2. Sequence diagram definition.
3. System state diagram definition.
4. Functions pseudocode development.
5. Functionalities development.

Realized activities as planned:

1. Application diagrams of specific activities definition.
2. Sequence diagram definition.
3. System state diagram definition.
4. Functions pseudocode development.
5. Functionalities development.

Realized activities no planned:

1. Class diagram modification.
2. Components diagram modification.
3. Application overall diagram of activities modification.

Activities for the next 2 weeks:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Progress Report # 5

Name: Adalberto Cubillo Rojas.
Weeks: from 14 of November to 25 of November 2011.

Planned activities:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Realized activities as planned:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.

Realized activities no planned:

1. Modifications on the components diagram.
2. Modifications on the class diagram.

Pending activities for the next 2 weeks:

1. Optimization testing.

Activities for the next 2 weeks:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Progress Report # 6

Name: Adalberto Cubillo Rojas.
Weeks: from 28 of November to 09 of December 2011.

Planned activities:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Realized activities as planned:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Realized activities no planned:

1. Add extra functionalities to the project.
2. Extra functionalities testing.

Activities for the next 2 weeks:

1. Functions pseudocode development.
2. Functionalities implementation.
3. Functionalities testing.
4. Optimization testing.

Progress Report # 7

Name: Adalberto Cubillo Rojas.
Weeks: from 12 to 23 of December 2011.

Planned activities:

1. Functionalities testing.
2. Optimization testing.

Realized activities as planned:

1. Functionalities testing.
2. Optimization testing.

Realized activities no planned:

1. Research code optimization rules in Java.

Activities for the next 2 weeks:

1. Code optimization.
2. Functionalities testing.
3. Optimization testing.

Progress Report # 8

Name: Adalberto Cubillo Rojas.
Weeks: from 2 to 13 of January 2011.

Planned activities:

1. Functionalities testing.
2. Optimization testing.
3. Code optimization.

Realized activities as planned:

1. Functionalities testing.
2. Optimization testing.
3. Code optimization.

Realized activities no planned:

1. Modifications on the graphical user interface.
2. Modifications in the functionalities.
3. Presentation of the project.

Activities for the next week:

1. Documentation of the project.
2. Deployment of the project.